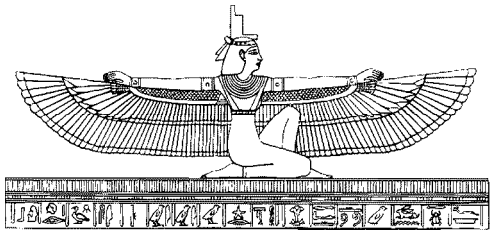


Papyrus Objects Interfaces
Receivers and Type Managers
Administrator's and Developer's Guide

Document number precdae/620/2

ISIS Information Systems GmbH
Alter Wienerweg 12
A-2344 Maria Enzersdorf/Vienna
Tel: +43 (0) 2236 27551 0*
Fax: +43 (0) 2236 21081
Product Support: +43 (0) 2236 27551 111
support@isis-papyrus.com
General Information: info@isis-papyrus.com
Website: www.isis-papyrus.com

© **Copyright ISIS Papyrus Software AG 2006**

All rights worldwide are the property of
ISIS Papyrus Software AG.
Changes to the Software, Help Texts and Manual may
be made without prior announcement.

Reproduction in whole or in part, of this manual,
or of the information on the accompanying media,
is only permitted with the express permission of
ISIS Information Systems GmbH.
Excepted from this rule are legally and contractually
bound licencees of **ISIS** Papyrus Software AG.

This document is available also in PDF format converted
with the ISIS Papyrus PDF-Server.

Table of Contents

1.0 Overview to the Papyrus Objects manuals **1**

2.0 Typographic Conventions **3**

3.0 Introduction **4**
3.1 Adapters & Receivers 4
3.1.1 Adapter and Agent interaction 5

4.0 Adapter **6**
4.1 States 7
4.2 Methods 8
4.3 Attributes 9
4.4 Adapter using Match Criteria 10
4.4.1 Types of Match Criteria 11
4.4.1.1 Task and its children 12
4.4.1.2 Task and Queue 12
4.4.1.3 Queues 12
4.4.1.4 User object 13
4.4.2 Overall process of an Adapter setup utilizing Match Criteria 15
4.5 Adapter using PQL Rules - Factory Features 15

5.0 Scheduler **16**
5.1 States 18
5.2 Methods 18
5.3 Attributes 19

6.0 Receiver Messages **20**
6.1 Receiver Message Container 20
6.1.1 States 20
6.1.2 Methods 20
6.1.3 Attributes 20
6.2 Receiver Message objects 21
6.2.1 Attributes 22
6.2.2 Label the Queue with an additional Match Criterion 23

7.0 File Receiver (Dispatcher, ICD Receiver) **24**
7.1 File Receiver Configuration 25
7.1.1 Profile Description of the dispatch.ini 25
7.1.1.1 <General> Parameter Summary 25
7.1.1.2 <Control_section> Parameter Summary 26
7.1.1.3 <FixResource> Parameter Summary 27
7.1.1.4 <File_control> Parameter Summary 27
7.1.2 dispatch.ini - Sample Profile 35
7.1.3 Starting the File Receiver 35
7.1.4 File Receiver and Papyrus Host 36
7.2 Usage of the File Receiver with Papyrus Server 40
7.2.1 ICD Parameter File 40
7.2.1.1 [PPMF] Parameter Summary 40
7.2.1.2 Parameter description of the PPMF section parameters 52
7.2.2 Parameter Summary for the Modules 52
7.3 Usage of the File Receiver in Papyrus WebControl 52
7.3.1 Mapping ICD parameters to attributes 53
7.3.2 Sample setup of a File Receiver 56

8.0 PQL Based File Mapping **57**
8.1 Configuration of the PQL Based File Mapping 57
8.1.1 Adapter Setup 57
8.1.2 FM Group and FM Entry objects 58
8.1.3 PQL Scripts 61

8.2 Usage of the PQL Based File Mapping 61

9.0 XML Receiver 62
9.1 Components of the XML Receiver 62
9.1.1 Attributes of the class "XML Receiver" 66
9.1.2 Instantiation of the appropriate Task template via Match Criteria 67
9.1.3 Mapping XML data into attributes 68
9.1.4 Predefined Entities 69
9.2 Usage of the XML Receiver 72

10.0 Fax Receiver / Sender 73
10.1 Hardware installation and configuration 74
10.1.1 Determine the COM port 75
10.1.2 Determine the fax class of the modem 76
10.1.3 Identify the model 76
10.1.4 Supported modems 77
10.1.5 Flow control settings 79
10.1.5.1 Hardware flow control 79
10.1.5.2 Software flow control 80
10.1.6 Fax hang-up error list 81
10.1.7 Fax Standards and Protocols 82
10.1.7.1 Group 3 82
10.1.7.2 V.27ter 82
10.1.7.3 V.29 82
10.1.7.4 V.17 82
10.1.7.5 Class 1 & class 2 83
10.2 Components of the Fax Receiver / Sender 83
10.2.1 Attributes of the base class "SERIAL RECEIVER" 84
10.2.2 Attributes of the class "FAX RECEIVER" 86
10.2.3 Attributes of the class "FAX MODEM" 86
10.2.4 Attributes of the class "FAX" 87
10.2.5 Attributes of the Tool class "FAX TIFFG3" 87
10.2.6 tool/material method of Tool class "FAX TIFFG3" and Material class "FAX AFP" 87
10.2.7 tool/material method of class "FAX RECEIVER" and Material class "FAX AFP" 87
10.3 Usage of the Fax receiver sample 88
10.3.1 Modem initialization 88
10.3.2 Receiving of fax messages 91
10.3.3 Troubleshooting 91
10.3.3.1 Phase A 91
10.3.3.2 Phase B 91
10.3.3.3 Phase C 92
10.3.3.4 Phase D 92
10.3.3.5 Phase E 92
10.3.4 Instantiation of the appropriate Task template 92
10.3.5 Mapping Fax data into attributes 93
10.4 Usage of the Fax sender sample 94
10.4.1 Modem initialization 95
10.4.2 Sending of fax messages 97
10.4.3 Troubleshooting 97
10.4.3.1 Checking of TIFF G3 and attributes 97
10.4.3.2 Phase A 97
10.4.3.3 Phase B 98
10.4.3.4 Phase C 98
10.4.3.5 Phase D 98
10.4.3.6 Phase E 98
10.4.4 Using a fax number defined in the AFP 99
10.4.5 Instantiation of the appropriate Task template 101
10.4.6 Mapping Fax data into attributes 101
11.0 HTTP Receiver / Sender 102
11.1 Object Structure 103

11.2 Basic handling of the HTTP Receiver / Sender ... 103
11.3 Web Server Functionality ... 104
11.4 Receiving POST Data ... 105
11.5 Sending back a response ... 106
11.6 Sample HTTP Receiver ... 106
11.6.1 HTTP Receiver / Sender ... 106
11.6.2 General Container ... 108
11.6.3 Queue ... 108

12.0 PPSClient ... 109
12.1 Synopsis ... 109
12.2 Options ... 110

13.0 Output of externally generated print data streams ... 111

14.0 MQ Receiver and MQ Sender ... 115
14.1 MQ Receiver ... 116
14.1.1 States ... 116
14.1.2 Methods ... 117
14.1.3 Attributes ... 117
14.1.4 Setup of MQ Receiver ... 117
14.1.4.1 Match criterion for the instantiation ... 118
14.1.4.2 Match criteria to populate attributes with the received header data ... 118
14.1.4.3 Match criteria to populate attributes with the received data ... 119
14.1.4.4 Match criteria to populate attributes with the Message ID and the Correlation ID ... 120
14.2 MQ Sender ... 121
14.2.1 States ... 121
14.2.2 Methods ... 121
14.2.3 Attributes ... 124
14.2.4 Setup of MQ Sender ... 124
14.2.5 Definition of outbound messages ... 125
14.2.6 Analysis of inbound messages ... 126
14.2.7 Definition of the Material MQSendToPrintPool ... 127
14.3 Material "MQ send" ... 128
14.3.1 states ... 128
14.3.2 Methods ... 128
14.3.3 Attributes ... 129
14.4 Using Client Connection Channels under MQ ... 129
14.4.1 Definition of a Client Connection Channel ... 131

15.0 POP3 Receiver ... 132
15.1 States ... 133
15.2 Attributes ... 133
15.3 Methods ... 133
15.4 Data transmitted to the Adapter ... 135
15.5 Data templates for the E-Mail template ... 136
15.6 Setup of the POP3 Receiver ... 136
15.7 Components of the POP3 Receiver ... 137
15.7.1 POP3 MAILBOX ... 137
15.7.1.1 States ... 137
15.7.1.2 Attributes ... 138
15.7.1.3 Methods ... 138
15.7.2 E-mail object ... 139
15.7.2.1 States ... 139
15.7.2.2 Attributes ... 141
15.7.2.3 Methods ... 141

16.0 SAP Receiver / Sender ... 142
16.1 Settings on SAP R/3 ... 144
16.1.1 SAP Easy Access ... 144

16.1.2 Spool Administration : Real Output Management System 147
16.1.3 Spool Administration : Logical Output Management System - SAP Configuration 148
16.1.4 Spool Administration : Logical Output Management System - OMS Configuration 149
16.1.5 Spool Administration : Output Device 150
16.1.6 Spool Administration : List of Operating Systems 151
16.1.7 Spool Administration : Operating Systems Commands 153
16.1.8 Transaction SP01 153
16.2 Setup on Papyrus Objects 155
16.2.1 Attributes of the SAP Factory Adapter and the SAP Adapter templates 156
16.2.2 Attributes of the SAP Receiver / Sender 158
16.2.3 Attributes of the SAP Task 161
16.2.4 Methods of the SAP Task 161

17.0 Host Client Receiver 162
17.1 Components of the Host Client Receiver 162
17.1.1 Attributes of the class "HOST RECEIVER" 163

18.0 Papyrus Type Manager DB 164
18.1 Type Manager DB for standard SQL databases 164
18.2 class Type Manager DB 166
18.3 Type Manager DB Example 166

19.0 Type Manager LDAP 167
19.1 Components of the Type Manager LDAP 169
19.1.1 Type Manager LDAP 169
19.1.1.1 States 170
19.1.1.2 Methods 170
19.1.1.3 Attributes 170
19.1.2 LDAP Query 171
19.1.2.1 States 171
19.1.2.2 Methods 171
19.1.2.3 Attributes 172
19.1.3 LDIF File 172
19.1.3.1 States 172
19.1.3.2 Methods 172
19.1.3.3 Attributes 172
19.1.4 Batch folder 173
19.1.5 User External 173
19.1.5.1 states 173
19.1.5.2 Methods 173
19.1.5.3 Attributes 174
19.1.6 Role External 174
19.1.6.1 states 175
19.1.6.2 Methods 175
19.1.6.3 Attributes 175
19.1.7 Policy External 175
19.1.7.1 States 175
19.1.7.2 Methods 175
19.1.7.3 Attributes 176
19.1.8 Group External 176
19.1.8.1 states 176
19.1.8.2 Methods 176
19.1.8.3 Attributes 177
19.2 Match Criteria for the Authentication templates 177
19.3 Setup of the Authentication templates 178
19.4 General setup inside the LDAP directory 181

Glossary 189

Index **189**

1.0 Overview to the Papyrus Objects manuals

The following table gives a short overview to all manuals related to Papyrus objects. Please refer to the document number in brackets below the title when inquiring for manuals.

Installation of Papyrus Objects for Papyrus WebRepository & WebControl *Installation and Administrator's Guide* `(poinste)`	This document describes the installation and configuration of Papyrus Objects which is the underlying infrastructure component of Papyrus WebControl and Papyrus WebRepository. Furthermore the creation and configuration of new Nodes and Papyrus Objects Kernel parameters are described. In addition the document covers issues like backup and recovery recommendations.
ISIS Papyrus WebRepository and Papyrus Objects *Developer's Guide* `(pwride)`	This document describes Papyrus WebRepository which is an object oriented graphical workflow management system. It is used to build highly flexible frameworks which allow to control arbitrary processes.
Print Management Framework for ISIS Papyrus WebControl *Administrator's Guide* `(pwciae)`	This document describes the Print Management Framework which is used for the print management across platforms. Papyrus WebControl is the underlying object oriented workflow management system for the Print Management Framework and provides a graphical user interface to control the Papyrus Server Modules.
Papyrus Correspondence Framework for ISIS Papyrus WebRepository *Administrator's and Developer's Guide* `(pcorade)`	This document describes the Correspondence Framework which is used for the system integration of distributed document applications.
Papyrus Objects Interfaces Receivers and Type Managers *Administrator's and Developer's Guide* `(precdae)`	This document describes all interfaces which enable the communication of Papyrus Objects with external applications and processes (e.g. MQ Series, SAP, etc.)

ISIS Papyrus WebPortal *Installation and Developer's Guide* (pwpade)	This document describes the thin client solution Papyrus WebPortal which allows to access the Object Space of a certain Node via a standard web browser.
Tuning Papyrus Objects Installations for Papyrus WebRepository & WebControl *Administrator's Guide* (potgae)	This document describes how the default setup of Papyrus objects can be customized to the system architecture and requirements of an organization to increase the overall performance.
ISIS Papyrus COM Agent Module *Reference Guide* (pwrcom)	This document describes the COM Agent which is an interface for Papyrus WebRepository and WebControl to receive data via a serial port.

2.0 Typographic Conventions

The following table describes the typographic conventions used in this manual.

Font style	Description	Example
Bold	Words to be emphasized	references are **Links** to objects.
Italic	Papyrus Objects terms	While *Action methods* are self contained, *Tool / Material methods* can only be executed in combination.
Bold Italic	Highlighted Papyrus Objects terms	The ***Queue*** must be a child of the ***Adapter***.
"Normal"	attribute and field names without object name	If the boolean expression in the field "Change Trigger" becomes true, the state transition is triggered.
"Normal / Normal / ..."	Paths (separated by slashes)	Open the Library Folder "Library / Print Server / Print Server Components / Miscellaneous Tools".
Italic.Italic	attribute and field names with object name	*Data.SendEvent* = allparents.Goto_Ready(self,x)
"Bold"	Dialog names	The **"objects Info Window"** shows information about the currently selected object.
"Bold | Bold | ..."	Menu items (separated by pipe symbols)	Via the menu item **"Options | Use MDI"** it is possible to toggle between MDI and SDI view.
"Italic"	Book titles and chapters	Refer to the general readme *"ISIS NetKey Support"* for further information.
`Fixed width`	Code segments and field values are written with fixed width.	`TotalAmount >= 2000000`

3.0 Introduction

This chapter gives a general introduction to Papyrus Objects Interfaces which allow to connect Papyrus Objects with the outside world (external applications and processes).

There are two types of Papyrus Objects Interfaces:

Adapter and Receiver

A Receiver is used to receive messages and events outside Papyrus Objects passively, it passes these messages and events over to the Adapter, which has definitions on how to react, to convert them into activities inside Papyrus Objects. In most cases, the activity is the need to instantiate a Task that will execute the required functions inside *Papyrus Objects*. The Adapter class can be used for any type of Receiver, the attribute "Type" of the Adapter is used in the field "Type Property" of the attributes of the Task templates and its children to define match criterions. Like any other user, the Adapter requires a Role which grants him authorization to do his job.

Type Manager

A Type Manager is an interface which can access data stored outside Papyrus Objects actively (c.f. Receiver is passive). Type Manager DB enables access to certain database products without SQL, Type Manager LDAP can be used to obtain user authentications defined outside Papyrus Objects.

Note: The ***Papyrus File Receiver (Dispatcher)*** makes it possible to continue to use existing ICD files with the new Papyrus Server architecture in *Papyrus Objects*. For new setups the ***XML Receiver*** is highly recommended as it is much more flexible.

3.1 Adapters & Receivers

If it is not required to access data outside Papyrus Objects actively an Adapter / Receiver setup should be used. The Adapter uses the data (e.g. name value pairs, XML data structure) passed over by the Receiver to instantiate Task templates and to fill attributes. To accomplish this **Rules or Match Criteria** have to be defined. By default the Adapter instantiates Task templates into its child Queue, nevertheless it is possible to let the Adapter instantiate Task templates into any other type of object. Like any user, the Adapter requires a Role which grants him authorization to perform its tasks.

3.1.1 Adapter and Agent interaction

The Receiver processes data from various sources (MQ, HTTP, ICD, XML, POP3, etc.) and sends the data to the Adapter using a general interface **(1)**.

The Adapter uses the data from the Receiver and instantiates a Task template based on a Match Criterion found in the field "Type Property" of an attribute or based on a Rule attached to the Task template with a keyword like $MATCH_LOCAL$ in the THEN clause **(2)**.

The Agent moves the Task to the proper Queue **(3a)** or runs the Tool / Material method of Tools that match Materials of the Task **(3b)**.

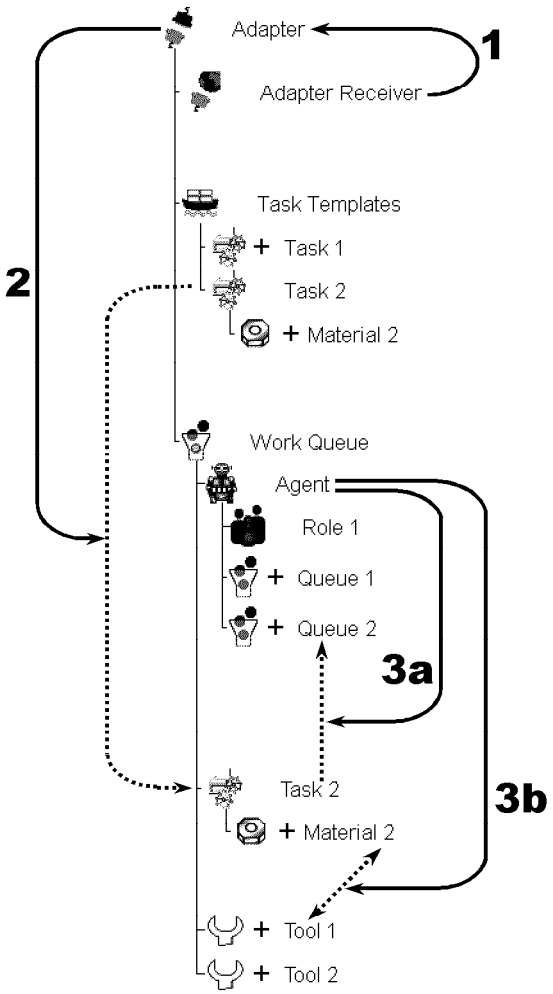


Figure 1: Adapter and Agent interaction

4.0 Adapter

An Adapter is used to process the data transmitted by the Receiver or data stored in an object tree dropped onto it. While there are several types of Receivers defined (XML, MQ, FAX, SAP, POP3, FILE, etc.) only one type of Adapter is required. To be operable only the attribute "Type" has to contain a string that is later used in Match Criteria to identify the Adapter. Based on Rules or Match Criteria the Adapter instantiates templates and fills attributes of the instantiated templates and their child objects with the data transmitted by the Receiver.

To enable the Adapter to instantiate templates based on the data transmitted by its Receiver, a General Container with appropriate templates (usually Task templates) is required below the Adapter. By default the Adapter instantiates its templates into its Queue. If there is more than one Queue referenced below the Adapter, it generates instances in all Queues, if no Queue is referenced below the Adpater, it uses the objects as destinations with a match criterion of type INPUT_SIZE. If no destination objects could be found, no instances are generated, nevertheless all other actions are performed (e.g. executing the PQL script of a PQL Rule). If the Queue referenced below the Adapter is located on another Node, a Message template is required in the General Container to take a note which instantiations are postponed until the required Nodes are online again. If the Adapter finds a matching instance instead of a template in its General Container, it generates a new reference of it in the destination object instead.

If an Adapter tries to instantiate a template or link an instance into a Queue (or any other destination object) that is closed as the limit of child objects defined via a match criterion with keyword INPUT_SIZE has already been reached, the Adapter rejects the job and sends "Confirmed_LinkRejected" (not an error like "Confirmed_Error") to its Receiver, which holds the job and resends it as soon as the Adapter sends "Confirmed_LinkReopend".

If only the factory features of the Adapter (processing of Rules instead of Match Criteria) are required, the Factory should be used, which is a simplified variant of the Adapter. Like the User object and the Agent, the Adapter requires a Role which grants him authorization to do its job and a Project in deployment state "Development" if versioned instances should be generated from templates (e.g. Rule templates). Add a Log object derived from the class "TOOL LOG" below the Adapter to record all messages of the Adapter and Receiver. Select the Log object and click the button "View log" to view the log or click the button "Clear log" to delete it.

If the Adapter should be used to instantiate templates that generate versioned instances (= instances inheriting the versioning methods with method internal and implementation name "ProjectNewVersion" or "ProjectNewVersionOfSelected"), add a reference of the Project that should be used to the Adapter, otherwise the Adapter cannot instantiate these templates if the user did not select a Project first via menu "File | Change Active Role and Project".

The additional system attributes **"DispatchFileName"** and **"DispatchFileExt"** are generated for each Task instance generated by the Adapter. They contain the filename and the extension of the input file (e.g. line data input file) and can be output in the visible name of the Task instance (e.g. visible name "Task - %DispatchFileName%").

4.1 States

Stopped	The Adapter is disabled and will not perform any tasks.
Started	The Adapter is enabled and will perform all operations.
Stopping	The Adapter is shutting down.
Error - starting	An error ocurred during startup of the Adapter.
Error	The Adapter is not operable due to an error, which can be determined via the Adapter / Receiver Log or the message panel (e.g. the Adapter has no appropriate Role attached or no Role at all).
Starting	The Adapter is starting up.
Auto-start on 'New'	If this state is defined as Initial state in a template or derived class, the Adapter is automatically started right after the instantiation as the state Transition "To -> Starting..." contains the Change trigger "true" which always evaluates to true.
Sleep mode	An Adapter in this state is used for backup reasons. As soon as a Node gets offline, the Adapter in state "Sleep mode" on the Backup Node takes over.
User intervention required	As soon as the original Node gets active again and thus the Backup Adapter is no longer required, the original Adapter changes to this state. Select the Adapter, click the button "Reset" and then "Start" to restart the original Adapter.
Busy	This state and the related state transitions are required if the Factory features of the Adapter are used, which is the case if the Adapter finds at least one Rule in the Task template or one of its child objects. The Program Event "BeginRunFactory" triggers the state transition "to -> Busy" (defined in the states "Stopped" and "Started") and the Program Event "EndRunFactory" triggers the state transition "to -> Started" (defined in the state "Busy").

4.2 Methods

Start adapter	Starts the Adapter which automatically starts the Receiver.
Stop adapter	Stops the Adapter which automatically stops the Receiver.
Delete reference	Overloaded with the expression EngineStatus==0 to ensure that an Adapter can only be deleted in state "Stopped" (state ID = 0).
Start adapter manually	Via this method the Adapter can be started manually by the user.
Move reference	Overloaded with the expression EngineStatus==0 to ensure that an Adapter can only be moved in state "Stopped" (state ID = 0).
Stop adapter manually	Via this method the Adapter can be stopped manually by the user.
View log	Displays the log file.
Show Adapter Information	Outputs general information about processes performed by the Adapter in the message panel including all attributes.
Start backup adapter	Via this Method An Adapter is brought into a standby mode if the parent object is a running Adapter.
Quick info	Same as method "Show Adapter Information" but only matches are shown without a complete list of attributes.
Run factory	This Tool method is used to process the data stored in object trees dropped onto the Factory Adapter. By dropping an object tree containing a Material derived from the class "RUN FACTORY", the tool/material method "Run factory" can be selected in the context menu to let the Factory Adapter process the attached Rule(s) using the object tree as input data.

All methods not listed above are deactivated and require no administration.

4.3 Attributes

Type

The attribute "Type" identifies the Adapter. Attributes containing a match criterion in the field "Type property" start with this name to specify which Adapter should be used.

Trace

If checked, additional tracing messages are output in the message panel and written into the log, if a Log object is attached to the Adapter. As the conversion of internal data to a human readable trace leads to a decrease in performance, this option should only be used for testing if problems occur.

Source Alias

A Source Alias only has be be defined if PQL Rules should be able to access the data stored in the object tree dropped on the Adapter (tool/material method "Run factory"). In all other cases no Source Alias is required and therefore the default expression **true;** should be entered as a placeholder performing no operations. If the Adapter for instance processes data transmitted by the Receiver (e.g. incoming XML files), the Adapter automatically generates a PQL set with the incoming data in variable "source". By declaring this variable in the PQL script (e.g. Dim source;) the incoming data can be accessed.

Example:
In the example below a PQL Rule is able to output the Object ID of the object dropped on the Adapter as the Source Alias @Inputobject is defined accordingly:

attribute "Source Alias" of Factory:

```
Dim @Inputobject;
@Inputobject=$self.$id, *;
```

attribute "THEN" of PQL Rule:

```
Dim @Inputobject;
Print("Object ID of input object: ", @Inputobject.$id);
```

Instance Alias

To enable Rules to access the object tree instantiated by the Adapter (from one of the templates in its General Container), at least one instance Alias has to be defined (e.g. if an Instance Alias is defined, the function newreference() in the THEN clause of a PQL Rule can be used to add a reference to a Task instantiated by the Adapter).

Template Alias

To enable Rules to access the object tree of templates in the General Container of the Adapter, at least one Template Alias has to be defined (e.g. if the XML tag <LockPrinter> contains the string "Yes" and a Template Alias is defined an expression in a PQL Rule can be used to change the field "Default" of an attribute to redirect all further print jobs).

Type	The attribute "Type" identifies the Adapter. Attributes containing a match criterion in the field "Type property" start with this name to specify which Adapter should be used.
Adapter Alias	Contains the object reference of the Adapter itself. By this the Adapter can be identified in the PQL script of the Rule (e.g. to determine the Node on which the Adapter is running via the line dim @Adapter=$self.$node;).
Analyze Blob	Has to be checked if XML files sent via MQ protocol have to be processed. With this option checked, PQL Rules can be used by the Adapter to navigate in the XML data structure sent as a BLOB by the MQ Receiver. Note that some parameters which are sent from the MQ Receiver to its Adapter are lost by this (e.g. CORREL_ID MESSAGE_ID).
Restart attempts	Specifies how many times the Adapter should attempt to restart if it stops because of an error.
Attempted restarts	Shows the number of restart attempts which have already been performed.
Restart interval [s]	The interval (in seconds) between an error and the next restart attempt.
Add job data to XML	In case the Adapter receives XML data, either by the XML Receiver or by the MQ Receiver (option "Analyze Blob" of the Adapter has to be checked) and the Receiver sends name/value pairs as well, the Adapter adds the job information as XML tags to the XML data, enabling a PQL script (e.g. in a Rule) to navigate and use this information like the rest of the XML tags.
Only type property	If selected only the match criteria in the Type properties are used and if there are Rules, they are ignored by the Adapter.

Note: If one of the Alias attributes is not required, at least the expression **true;** should be entered, otherwise the warning message POAD0134W appears.

All attributes not listed above are deactivated and require no administration.

4.4 Adapter using Match Criteria

To let the Adapter decide which Task templates of its General Container should be instantiated a Match Criterion containing one of the keywords **$MATCH_RECURSIVE$, $MATCH_LOCAL$ or $MATCH_ITERATION$** has to be defined in the field "Type property" of an attribute of the Task template or one of its children. When instantiating a Task template, the Adapter writes the values transmitted by the Receiver into those attributes of the Task and its children (Materials which contain Data objects and / or Formatter Resources) whose field "Type Property" contains a Match Criterion of type **AdapterType(Parameter1, Parameter2,...)**.

4.4.1 Types of Match Criteria

The "Type property" field of any attribute can be used to define one or more Match Criteria, separated by semi colons. The Adapter uses these Match Criteria to instantiate the appropriate Task template and to fill the attributes of new Task instance and its chid objects with the values transmitted by the Receiver. To let the Adapter perform more complex tasks (e.g. add child objects to the Task based on the input data, use attributes of an object dropped on the Adapter as input data, etc.) use the (see chapter "Adapter using Rules - factory features".

The following Match Criteria are sorted by location, i.e. in most cases the Match Criterion `AdapterType($MATCH_DESTINATION$,Parameter1, Parameter2, ...)` for instance will be defined in Queues. The placeholders **AdapterType and AgentType** in the following examples have to be exchanged by the string defined in the attribute "Type" of the Adapter or Agent. Via the attribute "Type" it is possible to identify an Adapter or Agent, thus a Match Criterion starting with the string "MyAgent" for instance only applies to the Agent containing the string "MyAgent" in its attribute "Type".

Note: It is even possible to enter match criteria with user defined keywords in the field "Type property", to evaluate them via a DOCDEF. For this purpose, make sure that none of the ISIS reserved keywords below are used, as this would cause unwanted side effects.

4.4.1.1 Task and its children

`AdapterType($MATCH_RECURSIVE$,Parameter)`

If the keyword $MATCH_RECURSIVE$ is used, the match criterion is **used by the Adapter to instantiate the template along with all its children** from the available templates (e.g. `ICD($MATCH_RECURSIVE$,QueueManagerQueueName)` to find the appropriate Task template for an ICD file via the ICD parameter QueueManagerQueueName).

`AdapterType($MATCH_LOCAL$,Parameter)`

If the keyword $MATCH_LOCAL$ is used, the match criterion is **used by the Adapter to instantiate the template without its children,** i.e. only those children which also have a match criterion defined that evaluates to true will be instantiated (e.g. `ICD($MATCH_LOCAL$,QueueManagerQueueName)` to find the appropriate Task template for an ICD file via the ICD parameter QueueManagerQueueName).

`AdapterType($MATCH_ITERATION$,Parameter)`

The keyword $MATCH_ITERATION$ extends the functionality of the keyword $MATCH_LOCAL$, it **enables the Adapter to instantiate each matching child object of this template repeatedly** and not only once. The child objects themselfes have to have match criteria defined to get instantiated, but due to the parent match criterion with keyword $MATCH_ITERATION$, they not only get instantiated once but as often as required (e.g. the matching PDF Data template gets instantiated three times by a POP3 Adapter as the incoming e-mail has three PDF attachments).

`AdapterType($SAVE_BLOB$)`

If the keyword $SAVE_BLOB$ is used, the match criterion is **used by the Adapter of an MQ Receiver to fill the attribute with the BLOB received by MQ**. If another Receiver is used, **the Adapter fills the attribute with the name / value pairs generated by the Receiver**.

`AdapterType(Parameter1, Parameter2, ...)`

If no keyword is used, the match criterion is **used by the Adapter to fill the attributes with the appropriate value(s)** from the input data (e.g. `ICD(PPMFInputName)` to map the ICD parameter PPMFInputName into the attribute "Line data name"). If more than one parameter should be mapped to the same attribute the additional parameter names are separated by commas (e.g. `ICD(AFPType,DEAFPType)` to map the ICD parameter `AFPType` or `DEAFPType` from an ICD file for a PageEXEC or a DocEXEC job into the attribute "AFP type" of the object "AFPDS").

4.4.1.2 Task and Queue

`AgentType(MUST_MATCH) and AgentType(CAN_MATCH)`

If the keyword MUST_MATCH or CAN_MATCH is used, the match criterion is **used by the Agent to move Tasks** into the appropriate Queues and **as a constraint for the execution of Tool / Material methods**, i.e. only if the corresponding attributes **(**= same internal name) of Tool and Material contain the same value the matching Tool / Material method will be started. With keyword MUST_MATCH, as the name indicates, all compared values have to be identical. Keyword CAN_MATCH only makes sense if there are more than one match criteria of this type defined, i.e. the Agent moves the object or allows the execution of a Tool/Material method of the object pair where most of the values match. As keyword MUST_MATCH can also be used to constrain the execution of Tool/Material methods, use keyword **MUST_MATCH_QUEUE** instead, to define that the match criterion should only be used to move Tasks. As keyword MUST_MATCH can also be used to move Tasks, use keyword **MUST_MATCH_TOOL** instead, to define that the match criterion should only be used to constrain the execution of Tool/Material methods.

Adapter`Type(INPUT_SIZE) or` Agent`Type(INPUT_SIZE)`

An attribute with this Match Criterion is used to define the **maximum number of child objects an Adapter or Agent may add** to the object with this attribute (if set to zero, there is no limit). By defining an attribute with the expression XML(INPUT_SIZE) in a Queue and setting it to 100 for instance, an XML Adapter (in this case with the string "XML" in its attribute "Type") cannot instantiate more than 100 Task templates into this Queue. By defining an attribute with the expression AGENT(INPUT_SIZE) in a Queue and setting it to 50 for instance, an Agent (in this case with the string "AGENT" in its attribute "Type") cannot move more than 50 Task templates into this Queue.

4.4.1.3 Queues

`AdapterType($MATCH_DESTINATION$, Parameter1, Parameter2, ...)`

A Match Criterion with the keyword $MATCH_DESTINATION$ is used as an identifier for Queues, it **enables the Adapter to find the appropriate Queue into which the template should be instantiated**. By default the Adapter instantiates templates into all child Queues, if a Queue contains such a Match Criterion the Adapter instantiates the template into this Queue only if the Match Criterion evaluates to true (e.g. if the Queues have the match criterion MQ($MATCH_DESTINATION$,8,6) defined in the Type Property of the additional attribute "Move target", the Adapter checks if the 6 digits string at position 8 of the received data is equal to the attribute containing this Type Property). If one of the Queues of the Adapter contains a match criterion of type $MATCH_DESTINATION$, a template of class "RECEIVER MESSAGE" has to be added to the General Container holding the Task templates of the Adapter. Add a further General Container (e.g. name it "Receiver Message Container"), thus the Adapter is able to create instances of this Receiver Message template in it. For a detailed description see chapter *"Receiver Messages"* below.

4.4.1.4 User object

`AdapterType($ISIS_objectNAME$, Parameter1)`

If the keyword **`$ISIS_objectNAME$`** is used, the match criterion is **used by the Adapter to fill the "Visible Name" of the object with the value** from the input data. The match criterion `XML($ISIS_objectNAME$,USEROBJNAME)` in the field "Type Property" of an additional attribute of a User template for instance can be used to map the contents of the tag <USEROBJNAME>JohnDoe<USEROBJNAME> of a XML file into the Visible name of the User object (Creation of a User via the XML-Receiver).

Note This is the only way to change the Visible name of an object automatically. A User containing a reference to an attribute, like %USER%, in the Visible name is invalid as the variable will not be resolved at logon.

`AdapterType($ISIS_object_INTERNAL_NAME$, Parameter1)`

If the keyword **`$ISIS_object_INTERNAL_NAME$`** is used, the match criterion is **used by the Adapter to fill the "Internal name" of the object with the value** from the input data.

4.4.2 Overall process of an Adapter setup utilizing Match Criteria

The Receiver takes the incoming data (e.g. it reads an ICD file), converts it (e.g. incoming e-mail is converted into a XML data structure) and transfers it to the Adapter. There is no other communication than this with the Adapter. Data is passed as a list of name value pairs (e.g. ICD Receiver) or as a XML data structure (e.g. POP3 Receiver, XML Receiver) to the Adapter. The Adapter must have a General Container among its child objects. In the Task templates of this General Container the Adapter looks for attributes whose "Type Property" fields contain a match criterion like ICD($MATCH_RECURSIVE$, ICD Parameter name).

If the contents of the attribute and the value of the parameter in the data transmitted by the Receiver are equal, the Task template will be instantiated by the Adapter. Multiple templates can be instantiated simultaneously for one job that was passed from the Receiver to the Adapter.

If the keyword **$MATCH_RECURSIVE$** is used the **Adapter instantiates** the **Task along with all its children** (the complete object tree), even children whose Match Criterions do not evaluate to true. If the keyword **$MATCH_LOCAL$ or $MATCH_ITERATION$** is used, **only the Task itself gets instantiated**. However, the Adapter continues to search for Match Criterions in the children of the Task. Whenever it finds another matching object, this object (and potentially its children) is instantiated as well as a part of the Task. Again the distinction between $MATCH_RECURSIVE$ and $MATCH_LOCAL$ / $MATCH_ITERATION$ applies here.

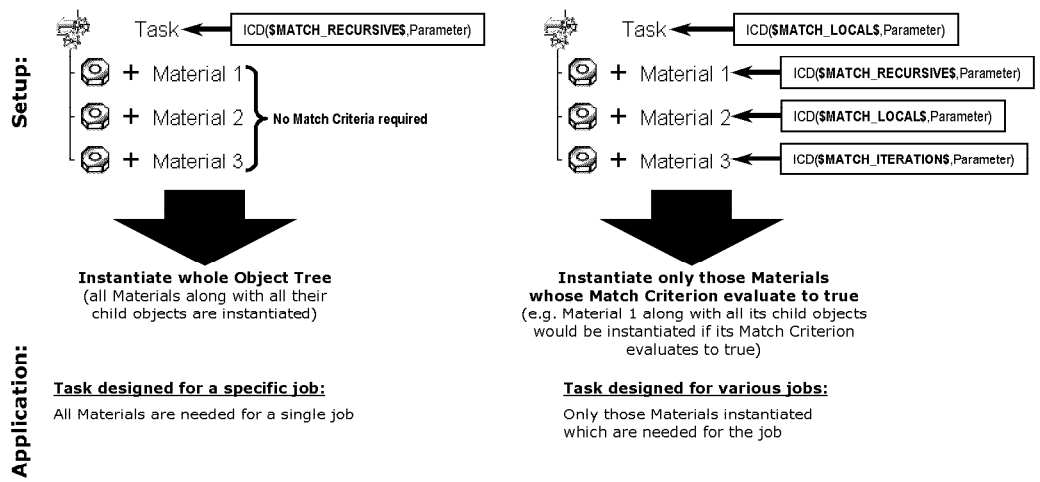

Setup:
Task
ICD($MATCH_RECURSIVE$,Parameter)
+ Material 1
+ Material 2
+ Material 3
No Match Criteria required
Instantiate whole Object Tree
(all Materials along with all their child objects are instantiated)
Task designed for a specific job:
All Materials are needed for a single job
Task
ICD($MATCH_LOCAL$,Parameter)
+ Material 1
ICD($MATCH_RECURSIVE$,Parameter)
+ Material 2
ICD($MATCH_LOCAL$,Parameter)
+ Material 3
ICD($MATCH_ITERATIONS,Parameter)
Instantiate only those Materials whose Match Criterion evaluate to true
(e.g. Material 1 along with all its child objects would be instantiated if its Match Criterion evaluates to true)
Task designed for various jobs:
Only those Materials instantiated which are needed for the job
Application:

Figure 2: Difference between $MATCH_RECURSIVE$ and $MATCH_LOCAL$

Note: This type of Match Criterion can be applied not only to Tasks and Materials but any other object as well!

During the process of instantiation, the parameter values that came from the Receiver are written to the appropriate attributes of the objects that are created if the attributes are defined accordingly.

Tool objects contain static parameters that do not change for each specific job, such as the IP address of a printer. Attributes that are likely to change for each job are contributed by the Material and are mapped to the Tool as described in chapter *"Print Management Framework"*. Usually Tools are already present in the Print Queues and do not need to be instantiated by the Adapter.

Via the method **"Show Adapter Information"** it is possible to show additional information retrieved by the Adapter in the Message window.

4.5 Adapter using PQL Rules - Factory Features

PQL Rules (class "PQL RULE" derived from base class "AGENT RULE") and PQL Match Rules (class "PQL MATCH RULE" derived from base class "AGENT RULE") can be used like Match Criteria to let the Adapter decide which Task template or child of a Task template should be instantiated (PQL Match Rule) and to fill attributes with the values transmitted by the Receiver (PQL Rule). The transaction is finished as soon as the PQL Match Rule and the PQL Rule have been successfully processed, until then the instance generated by the Adapter from one of its templates is not yet visible. The Adapter uses Rules in the following sequence:

1. The Adapter evaluates the condition in the WHEN clause of the PQL Match Rule below each template to determine which template(s) should be instantiated.
2. In the course of instantiation, the Adapter evaluates the condition in the WHEN clause of the PQL Rule(s) below the instance(s) to be generated.
3. The PQL Rule(s) below the instance(s) generated by the Adapter are exectued if they evaluated to true.

If the Adapter finds at least one Rule object attached to the Task or one of its child objects, it ignores the match criteria defined in the field "Type property" of the attributes and the factory features of the Adapter are used, i.e. the PQL statements of the Rules are used instead to instantiate the appropriate Task template and to populate the new Task instance and its child objects with the data transmitted by the Receiver. To override this default behaviour and define that match criteria should be used even if there are Rule objects in the Task template(s), select option "Only type property" in the Adapter.

By dropping an object tree containing a Material derived from the class "RUN FACTORY", the tool/material method "Run factory" can be selected in the context menu to let the Factory Adapter process the attached Rule(s) using the object tree as input data.

When an Adapter instantiates one of its templates, the Rule instances attached to this template are automatically removed in the generated instance. If the templates used for instantiation by the Adapter contain Rule templates, a Default Project has to be defined for the Adapter by referening a Project in deployment state "Development" below it. As instances derived from the class "AGENT RULE" are versioned the Adapter otherwise would not be able to instantiate the Rule templates.

If one Rule should be used in several Tasks, create additional references of this Rule. Changing one of these references automatically changes all others, which simplifies maintenance.

The **P**apyrus **Q**uery **L**anguage (PQL) can be defined in the WHEN and THEN clause of QL Rules and PQL Match Rules (see chapter "PQL - Papyrus Query Language") to navigate object trees or data structures, to read or modify attributes and their attribute Properties, and to apply methods to certain objects. While the Rules of an Agent are located in its General Container, the Rules utilized by an Adapter are attached to the Task templates and their child objects processed by the Adapter. A Rule processed by the Adapter is applied to the object it is attached to, i.e. it is the entry point.

Details about PQL can be found in chapter *"PQL - Papyrus Query Language"* of the manual *"ISIS Papyrus WebRepository and Papyrus Objects - Developer's Guide (pwride)"*.

5.0 Scheduler

The Scheduler object is used to manage all time dependent state transitions (field "Event type" set to option "Relative time" or "Absolute time"). All objects which are currently in a state that has a time dependent state Transition defined are called waiting objects. The system automatically adds references of these waiting objects (e.g. Agent, Adapter, Tool, etc.) to the Scheduler for monitoring.

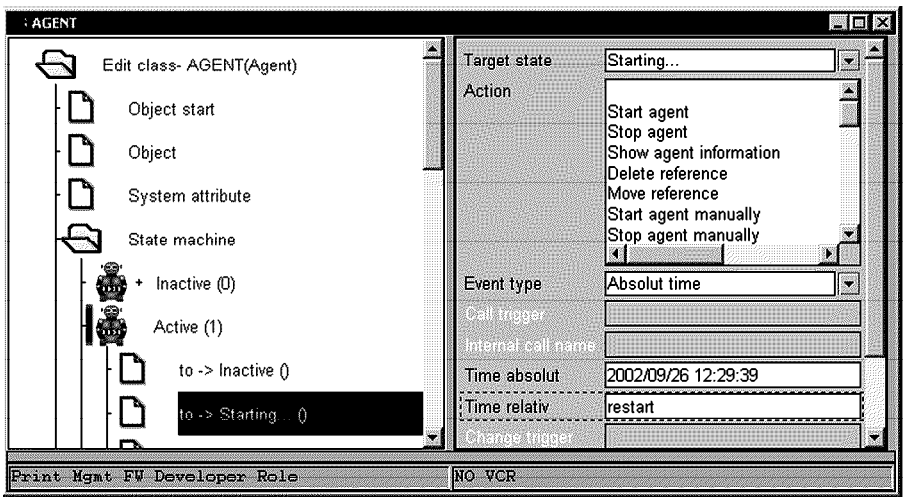
If an object is in an active state, which reflects that a related process is currently running, and the Papyrus Objects Kernel is shutting down, the related process will be stopped as well. To ensure that at start up of the Papyrus Objects Kernel, all related processes are started again, a state Transition from the active state has to be defined with the field "Event type" set to option "Absolute time" and the field "Time relative" containing the keyword "restart". As the Scheduler triggers these time dependent state Transitions on start up, the processes are automatically started.

Important!

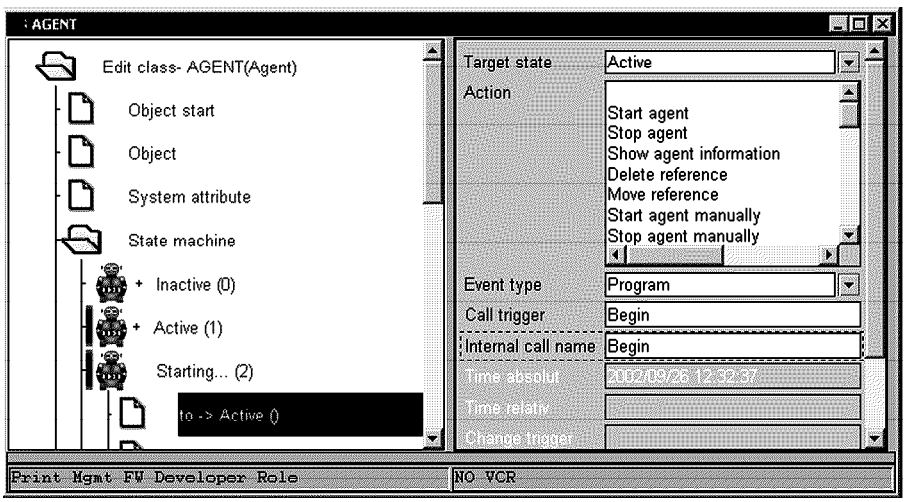
The maintenance of objects registered in a Scheduler of a certain Node demands for some Papyrus Objects Kernel resources of that Node which must not put too much strain on it. The Scheduler is designed to schedule a reasonable amount of objects on the Node owning the objects but must not be misused to schedule hundreds or more objects, which will cause serious performance problems. The developer has to take care in the class design of each object, that not too many time dependent state transitions that have to be monitored by the Scheduler are defined. This is especially true for frequently and by many processes created objects like Print Tasks, Letter Tasks, Capture Tasks, etc. If all of these Tasks would have to use the Scheduler, the Node owning these Tasks would be overloaded by hundreds of registered objects and could not perform normal work.

Example:

As the state "Active" of the class "AGENT" has the time dependent state transition "to -> Starting" defined, with the keyword "restart" in the field "Time relative", the Scheduler will trigger this state Transition at start up of the Papyrus Objects Kernel.



As soon as the state transition "to -> Active" in the state "Starting" receives the Program Event "Begin" by the DLL of the Agent, the state of the Agent changes back to "Active".



If the automatic starting of processes (e.g. caused by incorrect class definition) causes problems, it is possible to start the Papyrus Objects Kernel with the parameter "/Scheduler=No" to suppresses the start up of the Scheduler. By default, the last state of the Scheduler is used, i.e. if the Scheduler was started at the last shutdown it will be started the next time the Papyrus Objects Kernel is started (and vice versa).

5.1 States

Started (0)	The Scheduler is started to monitor waiting objects.
Stopped (1)	The Scheduler is stopped and will not monitor waiting objects.

5.2 Methods

Stop	Stops the Scheduler. This method is triggered via the state transition "to -> Stopped" defined in the state "Started", the Call trigger "Stop Scheduler" is displayed in the toolbar if the Scheduler is currently running. The method "Stop" is hidden as the checkbox "valid only for state machine" is selected in the method definition.
Start	Starts the Scheduler. This method is triggered via the state transition "to -> Started" defined in the state "Stopped", the Call trigger "Start Scheduler" is displayed in the toolbar if the Scheduler is currently stopped. The method "Start" is hidden as the checkbox "valid only for state machine" is selected in the method definition.
Waiting objects	Desktop View method to open a separate window, which displays all objects monitored by the Scheduler.
Check consistency	If the Scheduler got corrupt due to a problem in the file system (attributes "Waiting objects", "Waiting event", and "Waiting account" do not have the same number of entries), this method can be used to check and correct the consistency of information about the objects monitored by the Scheduler. This method can also be called via PQL (e.g. \0.0.0.1369.call("CheckConsistency"); to check and correct the Scheduler of the Domain Controller with Object ID = 0.0.0.1369).

All other methods are defined only to overload inherited methods. For these all Privileges have been deselected, to ensure that these methods cannot be applied to the Scheduler object.

Note: If the Scheduler got corrupt due to a problem in the file system (attributes "Waiting objects", "Waiting event", and "Waiting account" do not have the same number of entries), the original user account for the objects registered in the Scheduler might get lost and will be automatically replaced by the account used to start method "Check consistency", which is also performed during method "Check objects". If this account has no access to start the methods invoked by state transitions (e.g. Do activity = "Start agent" in state "Starting...", etc.) certain objects need to be manually restarted. To avoid such a situation, the System Administrator, or any other user who is in charge to start methods "Check objects" and "Check consistency", should have access to all methods that are invoked via time dependend state transitions or via the states resulting from them.

5.3 Attributes

Waiting objects

object ID of each waiting object. The following attributes are sorted in the same order.

Waiting event

The monitored state transition of each waiting object displayed as transition number. The first two bytes of the transition number indicate the state ID of the source state and the last two bytes indicate the position of the state transition (last defined = highest number), whereas both numberings start with 0. The transition number 0 is displayed if the related state Transition is performed automatically at start up of the Papyrus Objects Kernel.

Example:
By converting the transition number 65537 of the Agent (given in decimal format) into the hexadecimal number 0x00010001, the following information can be retrieved:
The first two bytes 0001 indicate that the current state of the Agent is the second defined state (state "Active", state ID = 1)
The last two bytes 0001 indicate that the second defined state transition is monitored by the Scheduler (state transition "to -> Starting", position number = 1).

Waiting for

Date and time for each waiting object, indicating when the state transition will occure. The transition time "1970/01/01 01:01:40" or "01.01.1970 01:01:40" is displayed if the state transition is performed automatically at start up of the Papyrus Objects Kernel.

6.0 Receiver Messages

6.1 Receiver Message Container

Since the Adapter may instantiate Task templates not only locally but also into Queues of remote Nodes it might happen that some of the remote Nodes are currently offline. In such a case the Adapter instantiates a matching Receiver Message template from its General Container (where the Task templates are located) into its Receiver Message Container to take a note which instantiations are postponed until the required Nodes are online again.

6.1.1 States

No state machine defined.

6.1.2 Methods

Contains inherited methods only.

6.1.3 Attributes

No attributes defined.

6.2 Receiver Message objects

Since the Adapter may instantiate Task templates not only locally but also into Queues of remote Nodes it might happen that some of the remote Nodes are currently offline. In such a case the Adapter instantiates a matching Receiver Message template from its General Container (where the Task templates are located) into its Receiver Message Container to take a note which instantiations are postponed until the required Nodes are online again.

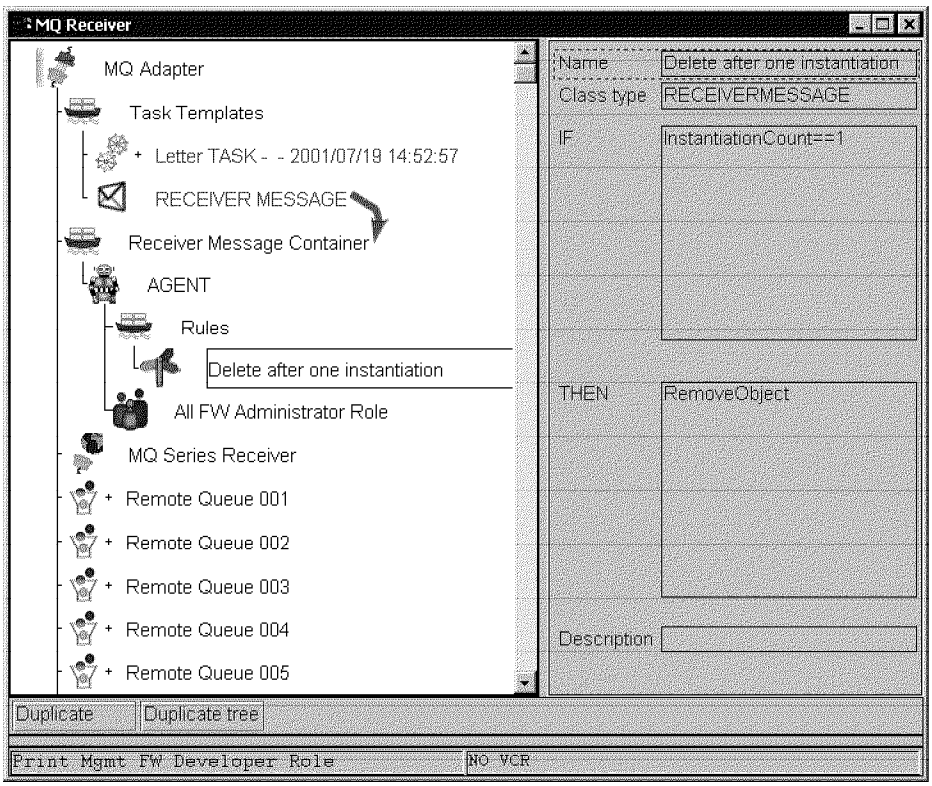


Figure 3: MQ Adapter setup to instantiate Task templates into remote Inboxes (Queues)

6.2.1 Attributes

All attributes are populated by the Receiver and should not be changed by the user:

Type ***(Type)***	Receiver writes file type in this attribute if a known type like JPEG, TIFF or GIF is stored in the Receiver Message object.
Destinations ***(Destinations)***	Object IDs of the remote Queues into which the Adapter should try to instantiate the Task templates (after becoming online again).
Parameters ***(Parameters)***	These attributes are used to temporarily store the data passed by the application as name/value pairs.
Blob ***(Blob)***	
Command ***(Command)***	This attribute contains the command that the Adapter will return to the Receiver on successful instantiation of the Receiver Message object.
Instantiation count ***(InstantiationCount)***	How many times the Task template has been instantiated successfully is stored in this attribute. If the Task should be instantiated only into one matching Queue, a Rule could be defined to delete the Receiver Message object if this attribute equals 1. If the Receiver Message object is not deleted, each time a Queue gets online the Adapter will use the Receiver Message object to check if an Instantiation is possible, thus for performance reasons a Rule should be defined to delete Receiver Message objects after a certain number of successful Instantiations.
Drop object ***(Dropobject)***	Object ID of the object which was dropped onto the Adapter.
Receiver ***(Receiver)***	Object ID of the Receiver that caused the Adapter to generate a Receiver Message object.
Creator ***(Builder)***	Object ID of the object that passed the message to the Adapter to generate a Receiver Message object (usually identical with Receiver).
Creation time ***(Buildtime)***	Creation time of the Receiver Message object, set by the application.
FileArray ***(FileArray)***	All file array data passed by the application.
MemArray ***(MemArray)***	All memory array data passed by the application.

6.2.2 Label the Queue with an additional Match Criterion

By default the Adapter instantiates Task templates into all child Queues, to let the Adapter decide into which Queue the Task template should be instantiated, the Queues requires an attribute (e.g. "Move target") containing a Match Criterion in the "Type Property" with the following syntax:

```
AdapterType($MATCH_DESTINATION$, Parameter1, Parameter2, ...)
```

(e.g. MQ($MATCH_DESTINATION$,8,6) checks if the 6 digits string at position 8 of the received data is equal to the attribute of the Queue)

7.0 File Receiver (Dispatcher, ICD Receiver)

The **File Receiver**, also known as Dispatcher or ICD Receiver, is used for the automation of predefined processes. It enables the Papyrus Server in combination with Papyrus WebControl to receive control files, which are either sent from the Host, using **Papyrus Host**, or from a workstation in the LAN. The control files can be transferred to the dispatch directory by any application. The dispatch directory will be monitored for incoming files, which are evaluated by the File Receiver and passed on to Papyrus WebControl for further processing. Furthermore the File Receiver makes it possible to continue to use existing ICD files with the new Papyrus Server architecture in Papyrus Objects. In order to make the transition to Papyrus WebControl as painless as possible, it is possible to use existing ICD files without any modifications. The File Receiver is fully compatible with existing ICD files and supports all parameters that can occur in ICD files.

The **Dispatcher** was an integral component of the Papyrus version 5.0x. From version 6 onwards it is still available as a component of Papyrus WebControl.

The following types of files can be processed

- ISIS Control Data files (*.ICD), processed as they are
- Papyrus Host job-profiles (*.CTL)
- Arbitrary input data files mapped to ICD files via the file mapping defined in the profile of the File Receiver (dispatch.ini) or using a file mapping based on PQL (for details see chapter "PQL Based File Mapping" below)

Relevant input formats for Papyrus Server are:

- AFP
- ASCII (General text files)
- Metacode (to be routed to a printer)
- Line data (for DocEXEC and PageExec)

Important!

- The **File Receiver** requires **space on the system's temporary directory**. If this is not accessible or out of space, Tasks will be instantiated properly but not contain the values from the ICD file parameters in their attributes which will cause errors in the continuing process flow.
- The **length of ICD parameters** may not exceede **1024 characters**.
- Under Unix it is not possible to determine if a file is still in use by a process. While on Windows operating systems the Dispatcher waits until the AFP is written by Papyrus DocEXEC, this is not possible on Unix operating systems. To figure out if a process is finished under Unix, the Dispatcher does not start processing until the size and timestamp of the file does not change for three polling intervals. In case the write operation is slower than the three polling intervals (default = 3x 200 ms), increase the polling interval via parameter **PollingInterval** of the profile dispatch.ini as described below. The best solution however would be to define a workflow that waits until the AFP is written and then moves it into the dispatch directory.

7.1 File Receiver Configuration

The File Receiver is installed in the directory **isis\pocxxVer\dispatch**, e.g. under Windows in isis\pocw3600\dispatch. The **configuration profile** (dispatch.ini) is located under **\isis\userisis.**

7.1.1 Profile Description of the dispatch.ini

The **dispatch.ini** contains all parameters for the initialization of the File Receiver functionality.

It is recommended to use the **dispatch.ini** as provided within the installation material and adapt it to meet specific requirements, e.g. the specification of paths.

Note:

The parameter names in the dispatch.ini are case sensitive and must remain unchanged.

Papyrus Desktop has to be restarted after changing dispatch.ini, otherwise the modifications have no effect!

The dispatch.ini consists of the following sections, described below:

- ***<General>***
- ***<Control_section>***
- ***<FixResource>***
- ***<File_control>***

7.1.1.1 <General> Parameter Summary

PollingInterval = Value

Specifies the interval (in milliseconds) in which the File Receiver scans the control directory for new files (default = 200 ms).

7.1.1.2 <Control_section> Parameter Summary

This section is used with **Papyrus Host** setups only.

Parameter Summary

```
APPCPROFILE = <Drive:>\Path\Filename     *on Unix: /path/filename
HostMode    = {Yes|No}
```

APPCPROFILE = <Drive:>\Path\Filename *(on Unix: /path/filename)

Specifies the file name and the path where the APPC profile is located.

This profile is used when CTL files are dispatched. It contains the alias and suffix definitions for the:

- Control library
- AFP resource library
- Input file library

Default: No default value.

A detailed description of the APPC profile can be found in *"Papyrus Host Installation and User's Guide"*.

HostMode = {Yes|No}

Specifies whether to use *Host Mode* or not. If switched on, it allows to run a certain job only once, i.e. a control file of a certain name will be dispatched only once. Another file with the same name will be ignored until the File Receiver is restarted.

This is useful when combining Papyrus Server with Papyrus Host transmitting data via SNA or TCP/IP which may cause an open-close-reopen sequence on the transferred data thus causing the same job being initiated several times.

Yes	The control file remains in the control directory until its complete processing.
No	The control file will be moved from the control directory to the log directory and deleted after the correct finish of the job (parameter **CtlBackUp** is set to **No),** or backed up **.** (See the chapter <File control>)

Default: Yes

7.1.1.3 <FixResource> Parameter Summary

This section is used with **Papyrus Host** setups only.

It describes the settings that are used for AFP resources, which are sent using the option **SENDRESOURCE=COMMON** or, **SENDRESOURCE=Yes** of the CTL-file.

Parameter Summary

```
FixResourceLBP  = <Drive:>\Path\Filename     *on Unix: /path/filename
FixFDFPath      = <Drive:>\Path              *on Unix: /path/
FixPDFPath      = <Drive:>\Path                  ---"---
FixResourcePath = <Drive:>\Path                  ---"---
```

FixResourceLBP = <Drive:>\Path\Filename

Specifies the location of the AFP resource library profile. See also the ***Default.lbp***

Default: No default value.

FixFDFPath = <Drive:>\Path

Specifies the path for the FORMDEF used for a PageEXEC run.

Default: No default value.

FixPDFPath = "<Drive:>\Path"

Specifies the path for the PAGEDEF used for a PageEXEC run.

Default: No default value.

FixResourcePath = <Drive:>\Path

Specifies the location of the resources, which are used for DocEXEC processing (e.g.: *.DFA and *.IDF files). All other resources are defined in the AFP resource library profile, which is specified by the **FixResourceLBP** parameter.

Default: No default value.

7.1.1.4 <File_control> Parameter Summary

Parameter Summary

```
AFPPRTLI           = <Drive:>\Path
AFPPELOG           = <Drive:>\Path
IDSFPATH           = <Drive:>\Path
ERRORPATH          = {Yes|No}
FileStructur       = #ICDFILE#<ICDParameterName>#...#
FileMapping        = $<SearchString>=#<ICDFileName>#<ICDParameterValue>#...#
CtlBackUp          = {Yes|No}
CtlBackUpDir       = <Drive:>\Path
UseBinaryattribute = {Yes|No}
```

AFPPRTLI = <Drive:>\Path

Specifies the control directory (path for incoming control files).

Do not move other files than these to this directory:
ISIS Control Data files (*.ICD)

Papyrus Host jobfiles (*.CTL)

Naming convention files (e.g. *.ASC, *.AFP)

Default: No default value.

AFPPELOG = <Drive:>\Path

Specifies the log path. If the parameter **HostMode=No**, all control and data files are moved into this directory after being dispatched and kept there for processing.

Related Parameter: HostMode

Default: No default value.

IDSFPATH = <Drive:>\Path

Specifies the path for the output files. Used only with **Papyrus Host**.

Default: No default value.

ERRORPATH = <Drive:>\Path

If an input file causes an error (e.g. it cannot be moved to the log directory because a file of the same name is already there and is currently being processed), it is moved to this directory.

Default: No default value.

FileStructur = #ICDFILE#<ICDParameterName>#...#

This parameter controls which ICD file will be used for a specific input file and which parameters are overwritten in these ICD files. It must always start with the string **"#ICDFILE#"**, which is a placeholder for the name of the respective ICD file to be used. These ICD files have to be present in the working directory of the File Receiver. If the **FileMapping** parameter is not defined, the File Receiver will load the associated ICD file and replace the ICD parameters defined in the FileStructur parameter with the values found in the input file name. If the **FileMapping** parameter is set, the File Receiver will load the associated ICD file and replace the ICD parameters listed in the **FileStructur** parameter with the values defined in the **FileMapping** parameter before passing the job to WebControl.

Example - obtaining the values from the input file name:

Assume that different input files should be automatically processed by calling either DocEXEC or the PCL module. The values to be used for the ICD parameters defined in the **FileStructur** parameter are taken from the names of the input files, i.e. these naming convention files contain the required ICD parameters separated by hashes in their filenames.

Path defined in AFPPELOG:

```
AFPPELOG=C:\ISIS\poc3600\dispatch\log
```

FileStructur parameter:

```
FileStructur=#ICDFILE#PPMFInputName#TargetName#PCL4LPRHost#DEDocDEF
        #DEEnvironment#
```

Input files to be processed (naming convention files):

```
#DE#thisfile#thisfile##MyDFA#$Env=val#.ASC
#PCL4#thisfile#thisfile#10.1.5.2###.AFP
```

ICD files in the File Receivers working directory (parameters to be overwritten are highlighted):

PCL4.ICD:

```
[PPMF]
  InputMode       = "AFP_AFP"
  TargetMode      = "AFP_PCL"
  PPMFInputName   = "<dummy>"
  TargetName      = "c:\isis\demo\output\pcl\<dummy>"
  PPMFLib         = "c:\isis\userisis\default.lbp"
  ...
[PCL]
  PCL4Portorqueue = "2"
  PCL4Lprprinter  = "PASS"
  PCL4Lprhost     = "<dummy>"




  ...
```

DE.ICD:

```
[PPMF]
  InputMode     = "DocEXEC"
  TargetMode    = "AFP_AFP"
  PPMFInputName = "<dummy>"
  TargetName    = "c:\isis\demo\output\afp\<dummy>"
  PPMFLib       = "c:\isis\userisis\default.lbp"
  ...
[DE]
  DESourceRes   = "DFA"
  DETargetRes   = "DFA"
  DEDocDef      = "<dummy>"
  DEEnvironment = "<dummy>"
  ...
```

Result:

In this example input files that start with the string "#PCL4#" will be processed with the ICD file "PCL4.ICD". Input files starting with the string "#DE#" will be processed with the ICD file "DE.ICD".

The values of the filenames are used to overwrite the parameters in the respective ICD files, which have to be present in the working directory of the File Receiver:

#PCL4#thisfile#thisfile#10.1.5.2###.AFP:

```
PPMFInputName = "C:\ISIS\poc3600\dispatch\log\
                 #PCL4#thisfile#thisfile#10.1.5.2###.AFP"

TargetName    = "c:\isis\demo\output\pcl\
                 #PCL4#thisfile#thisfile#10.1.5.2###.AFP"

PCL4LPRHost   = "10.1.5.2"
```

```
#DE#thisfile#thisfile##MyDFA#$Env=val#.ASC:

PPMFInputName = "C:\ISIS\poc3600\dispatch\log\
                 #DE#thisfile#thisfile##MyDFA#$Env=val#.ASC"

TargetName    = "c:\isis\demo\output\afp\
                 #DE#thisfile#thisfile##MyDFA#$Env=val#.ASC"

DEDocDef      = "MyDFA"

DEEnvironment = "$Env=val"
```

Note:

- For the parameter **PPMFInputName**, only the name of the input file is used, while the path is taken from the parameter **AFPPELOG** of dispatch.ini (HostMode=No) or from the parameter **AFPPRTLI** of dispatch.ini (HostMode=Yes).
- For the parameter TargetName, again only the name of the input file is used, while the path is taken from the ICD file.
- The string "ThisFile" is a special placeholder which is always replaced by the name (without path) of the input file.

FileMapping=$<SearchString>=#<ICDFileName>#<ICDParameterValue>#...#

FileMapping is a substitution-table, where all possible cases of input file categories (e.g. AFP, ASC, etc.) should be defined. The FileMapping parameter contains mapping definitions (conditions written as equations), these define which ICD file should be used for which type of input file and which values should be written in certain ICD parameters.

If the search string on the left side of the equation, marked with a starting $-sign, is found in the name of the input file (can be the extension or any other part of the file), the ICD file is used which is defined on the right side of the equation (filename without extension between the first two hashes right after the equal sign). The ICD parameter values that follow are used to overwrite the ICD parameters defined in the **FileStructur** parameter.

Several of these mapping definitions can be defined, separated by commas.

The placeholder "thisfile" causes the corresponding ICD parameter to be overwritten with the complete name (without path) of the input file.

Default: No default value.

Note:

- The ICD parameter PPMFInputName, if overwritten, takes the filename from the overruling mapping definition, while the path is taken from the value defined in the parameter AFPPELOG.
- The ICD parameter TargetName, if overwritten, takes the filename from the overruling mapping definition, while the path is not changed. This way, PPMFInputName and TargetName can contain the same filename (the input filename) but different paths to avoid a naming conflict.

Example - using mapping definitions of the FileMapping parameter:

Assume that different input files should be automatically processed by calling either DocEXEC or the PCL module and the parameter FileMapping should be used to change the ICD parameters. The same ICD files are used as in the previous example.

Path defined in AFPPELOG:

```
AFPPELOG=C:\ISIS\poc3600\dispatch\log
```

FileStructur parameter:

```
FileStructur = #ICDFILE#PPMFInputName#TargetName#PCLLPRHost#DocDEF
          #Environment#
```

FileMapping parameter:

```
FileMapping = $AnInput=#DE#Thisfile#Thisfile#dummy#MyDFA#$Env=val#,
          $AFP=#PCL4#Thisfile#Thisfile#10.1.5.2#dummy#dummy#
```

Input files to be processed:

```
Demo1.AFP
AnInputFile.ASC
```

Result:

In this example input files that contain the string "AFP" will be processed with the ICD-file "PCL4.ICD", input files containing the search string "AnInput" will be processed with the ICD file "DE.ICD".

The values in the mapping definitions of the parameter FileMapping are used to overwrite the parameters in the respective ICD files, which have to be present in the working directory of the File Receiver:

```
Demo1.AFP:
  PPMFInputName = "C:\ISIS\poc3600\dispatch\log\Demo1.AFP"
  TargetName    = "c:\isis\demo\output\afp\Demo1.AFP"
  PCL4LPRHost   = "10.1.5.2"


AnInputFile.ASC:
  PPMFInputName = "C:\ISIS\poc3600\dispatch\log\AnInputFile.ASC"
  TargetName    = "c:\isis\demo\output\pcl\AnInputFile.ASC"
  DEDocDef      = "MyDFA"
  DEEnvironment = "$Env=val"
```

Note: No linebreaks are allowed in the parameter specifications!

CtlBackUp = {Yes|No}

Yes	After successful job termination, the control file will be moved to the directory defined by **CtlBackUpDir**.
No	No backup will be done.

Default: No default value.

CtlBackUpDir = <Drive:>\Path

Enter the directory name for the backup of the control file.

Default: No default value.

UseBinaryattribute = {Yes|No}

Yes The ICD Receiver passes the contents of the input file as BLOB to the Adapter. An attribute like "AFPDS Name" of the AFPDS Data object retrieving the input file via match criterion ICD(PPMFInputName) must be of type "Binary", as the Adapter stores the BLOB in it.

No The ICD Receiver only passes path and filename of the input file to the Adapter. An attribute like "AFPDS Name" of the AFPDS Data object retrieving the path and filename of the input file via match criterion ICD(PPMFInputName) must be of type "String".

Default: No

Host parameters

The sample *dispatch.ini* delivered with the ISIS CD contains several parameters used when working with ISIS **Papyrus Host** only. These parameters define which fields of the host control file should be put into the *Job info text* parameter of the ICD file. The parameters are JobName, JobID, DDName, Forms, Copies, Lines, Date, Time, Programmer, DataSet, Device, Sysout.

7.1.2 dispatch.ini - Sample Profile

See an example of the **dispatch.ini** under **..\isis\userisis.**

7.1.3 Starting the File Receiver

The File Receiver will be started automatically via **Papyrus WebControl.** Depending on user rights it can be manually started and / or stopped.

For test purposes the File Receiver can also run *stand-alone* and send the dynamically composed ICD-files into a directory instead of directly sending to Papyrus Desktop/Papyrus WebControl.

For **Windows** execute the following command e.g.:

...\isis\pocw3600\dispatch\psew3dsp.exe /Destination=<path>

Destination defines the path where the ICD file is to be generated.

7.1.4 File Receiver and Papyrus Host

When Papyrus Host receives a print job from e.g. JES (e.g. from PRT107, which is defined to forward the job to Papyrus Host instead of sending the job directly to a printer) it sends the following files to the File Receiver:

- AFP or line data file (input file)
- Resources (optional)
- Host Control file (*.CTL)

Only if the transfer of the input file and the resources was successful, the control file is sent.

Details on the Papyrus Host connection with the partner program under the Server operating system can be found in *"Papyrus Host Installation & User's Guide"*.

The File Receiver dynamically composes an ICD file from the host control file (*.CTL), the "ppmf.ini", and the printer definition file (*.PMS), utilizing the configuration profile "dispatch.ini" and sends it to Papyrus WebControl.

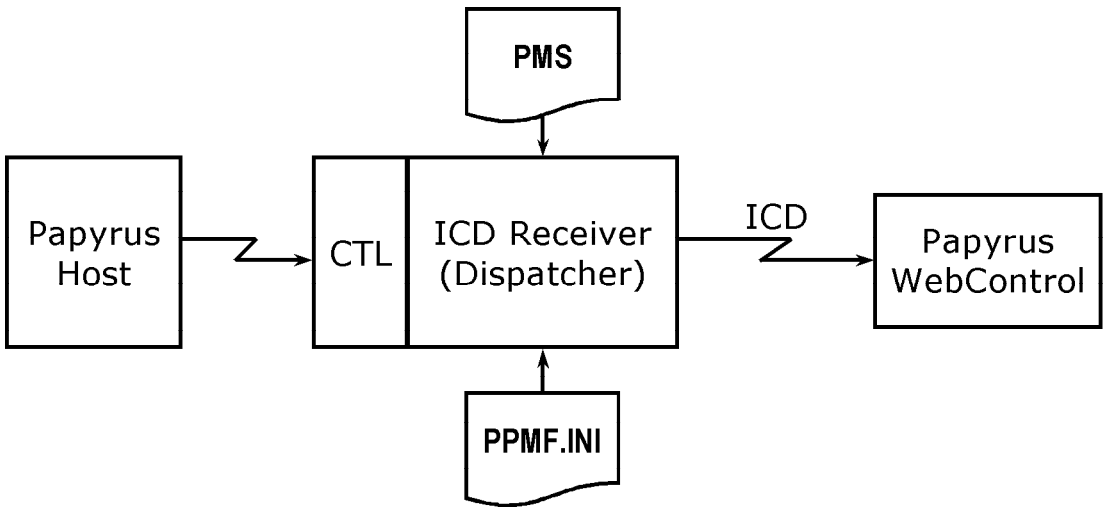


Figure 4: ICD receiver processing jobs sent by Papyrus Host

Control file (*.CTL)

The Control file contains the information which PMS-file should be used (e.g. if Device=PRT107 then PRT107.PMS is used), ICD parameters, and system parameters like the print class on the host (e.g. SYSOUT=3). The section <USERDATAParms> is intended for user defined parameters, any ICD parameters can be defined there as well. The parameter <Device> in section <JobIdentification> defines which PMS file has to be used.

Example for a control file:

```
<JobIdentification>
  Jobname=TCP10001
  JobID=JOBID
  UserID=CICSUSERTDA04CICS1PPHC0001
  ...

  Device=PRT107
  ...
<JesParms>
  Pagedef=ANY
  Formdef=ANY
  Prmode=LINE
  CC=YES
  TRC=NO
  ...

  Sysout=3
  ...
<USERDATAParms>
  USERDATA=InputMode=AFP_AFP
  USERDATA=PCL3NrOfCopies=3
  USERDATA=PCL3PortOrQueue=2
  USERDATA=PCL3LPRHost=10.1.7.57
  USERDATA=PCL3LPRPrinter=PASS
  USERDATA=MYPRINTER=LASERJET01
  ...
<Data>
  DataProfile=DAT107
  DataMember=fS4Tu2pt
  ...
<Resource>
  SendResource=NO
  ResourceProfile=RES107
<Control>
  ControlProfile= CTL107
  ControlMember=fS4Tu2pt
  ControlExtension=CNT
  FON         =RES107/<FON>
  CHS         =RES107/<CHS>
  CDP         =RES107/<CDP>
  ...
<End>
```

Printer definition file (*.PMS)

The PMS file contains parameters specific for the processing. PMS files are located in the File Receivers working directory. Depending on parameters in section <JesParms> of the CTL file (e.g. <Sysout=3>), several cases can be defined that decide which settings to use.

Example for a printer definition file:

```
<Sysout=3>
  Mode=PCL4
  Pcl4Fid=1
  Pcl4Mid=1
  Pcl4OffsetX=0
  Pcl4OffsetY=0
  Pcl4PortOrQueue=1
  Pcl4PrintQueue=LPT1
  Pcl4Remove=1
  Pcl4ToPrint=0

<FCB=STD>
  Mode=PCL3
  Pcl3Format=0
  Pcl3NrOfCopies=1
  Pcl3PrintQueue=LPT2
  Pcl3Remove=0
  Pcl3Resol=240
  Pcl3Sorted=0
  Pcl3ToPrint=1
  Pcl3UseFDF=1


<MYPRINTER=LASERJET01>
  Mode=PCL4
  Pcl4PortOrQueue=0
  Pcl4PrintQueue=\\URSA\LP
<End>
```

Note: If more than one condition is true (e.g. <Sysout=3> and <FCB=STD>) then the first one in the printer definition file is used.

Configuration profile "ppmf.ini"

The configuration profile "ppmf.ini" contains default settings that are used if neither the CTL-file nor the PMS-file have these parameters defined. It must be located in the File Receivers working directory.

Example for a ppmf.ini:

```
<JobControl_Default>
   Breaking=0
   CopyFrom=0
   CopyTo=0
   ...
<DocExec_Default>
   DEAcifIndex=No
   DEAfpLRecl=8192
   DEAfpType=PC
   ...
<PageExec_Default>
   AfpLrecl=4096
   AfpType=Pc
   ASCII=No
   ...
<End>
```

Configuration profile "dispatch.ini"

The configuration profile "dispatch.ini" contains all parameters for the initialization of the File Receiver functionality. Details can be found in chapter "File Receiver Configuration"

7.2 Usage of the File Receiver with Papyrus Server

Papyrus Server is organized modularly - its several modules are designed as separate executables.

To handle a print or output request, an ISIS Control Data file (ICD) with the required settings has to be defined.

Papyrus Server modules can be called directly with a single ICD parameter, either manually by entering the command line syntax or using a batch file or shell script.

When calling the module on the command prompt, ICD parameters can be also directly specified by their name and value prefixed by a forward slash, e.g. **/PPMFInputName=<.....>**.

Under \isis\demo\pse__icd several sample ICDs are provided, which demonstrate the use of the module parameter settings.

The directories \isis\demo\bat and \isis\demo\script... contain batch files and shell scripts that demonstrate the execution of Papyrus Server Modules using the provided ICD files on PCs and UNIX platforms.

See the following example of the command line syntax of an Papyrus Server MetaCode print request, assuming an installation under c:\isis:

```
c:\isis\psew3600\metacode\psew3met.exe
  /profile=c:\isis\demo\pse__icd\pc\pcaafp2meta.icd
```

The command has to be executed from the **\userisis** directory if relative paths in the used profiles are defined.

The parameter settings for the particular module are specified in the used ICD file.

Note: Every ICD file has to comprise a ***PPMF*** section containing general processing information. The available modules are listed in the manual *"ISIS Papyrus Server Installation and User's Guide"*.

7.2.1 ICD Parameter File

7.2.1.1 [PPMF] Parameter Summary

The [PPMF] section is common to all modules using ICD parameter files and needs to be specified for each processing mode when running the corresponding ISIS product.

The [PPMF] parameters comprise specifications concerning the general settings for each print or output request.

Note: Paths between double quotes are limited by default to 256 characters in ICDs and profiles.

See below a list of all parameters and their value options:

```
ppmf section:
PPMFInputName         = "\path\filename"  *on Unix:
                                                path/filename
TargetName            = "\path\filename"
InputMode             = "{AFP_AFP|AFP_ASCI|AFP_META|DOCEXEC|PE32}"
TargetMode            = "{AFP_AFP|AFP_FAX|AFP_FXG3|AFP_IJP|AFP_IPDS}"
                      = "{AFP_META|AFP_PCL|AFP_PCL3|AFP_PCL4|AFP_PDFT}
                      = "{AFP_PSCR|AFPTLECT}"
TargetType            = "{AFP_AFP|AFP_TIFF|AFP_GIF|AFP_PDF|AFP_PDFT}"
PPMFForm              = "filename without extension"
PPMFLib               = "\path\filename"
PPMFPPI               = "{240|300|600}"
AFPCodepage           = "number"
CodePagePath          = "\path\"           *on Unix: path/
DefaultFontLib        = "\path\filename"   *on Unix:
                                                path/filename
DefaultDBCSFont       = "string"
DefaultSBCSFont       = "string"
PageFrom              = "number"
PageTo                = "number"
CopyFrom              = "number"
CopyTo                = "number"

JobDisposition        = "{Release}"
FileDisposition       = "{Delete|Keep}"
QueueManagerQueueName = "string"
QueueManagerHandOver  = "{Yes|No}"
AbendOnError          = "{Yes|No}"
MaximumWarnings       = "0"
ReportMessageLevel    = "I"
StartBanner           = ""
EndBanner             = ""
JobInfoText           = "string"
JournalPath           = "\path"
MinFreeDiskSpace      = "number"
```

Note:
Not all parameters mentioned above are relevant for every type of job.

7.2.1.2 Parameter description of the PPMF section parameters

PPMFInputName

```
PPMFInputName = "<Drive:>\Path\Filename" *on Unix: /path/filename
```

Defines path and filename of the input file which will be processed (e.g.: line data, mixed AFPDS, or composed text AFPDS).

Default: No default value.

The attribute "Line data name (LineData_Name)" of the class "LINEDATA (FILE)", which is derived from the base class "DATA", has the expression *ICD(PPMFInputName)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter PPMFInputName.

TargetName

```
TargetName = "<Drive:>\Path\Filename" *on Unix /path/filename
```

Defines path and filename of the output file.

Default: No default value.

Each of the classes derived from the base class "DATA" with the string "(FILE)" at the end of their visible name (e.g. Attribute "PDF name (TargetName)" of the class "PDF (FILE)"), have one attribute defined with the Internal name "TargetName". The expression *ICD(TargetName)* in the field "Type property" of these attributes is used to let the Adapter fill this attribute with the contents of the ICD parameter TargetName.

InputMode

```
InputMode = "{AFP_AFP|AFP_ASCI|AFP_META|DOCEXEC|PE32}"
```

Specifies the input file format.

Default: AFP_AFP

The attribute "Input mode (InputMode)" of the class "MATERIAL" has the expression *ICD(InputMode)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter InputMode.

AFP_AFP :	Input file format AFPDS
AFP_ASCI:	Input file format ASCII
AFP_META:	Input file format Xerox Metacode This input file format can be used to route a Xerox Metacode file to a printer. In this case the output file format must be set as Xerox Metacode using **TargetMode=Afp_META** and the **MCOnOffline** parameter requires the value **1**. The PPMFLib parameter and the TargetName parameter will not be active, because the file is directly sent to the printer. Some Metacode parameters will also be ignored. The job details can be specified using the parameters beginning with the "MCXE" prefix.
DOCEXEC:	Input file format ASCII
PE32	Input file format ASCII

TargetMode

```
TargetMode = {AFP_AFP|AFP_FAX|AFP_PCL|AFP_FXG3|AFP_IJP|AFP_IPDS|AFP_META
            |AFP_PCL3|AFP_PCL4|AFP_PDFT|AFP_PSCR|AFPTLECT}
```

This parameter specifies the output file format.

Default: AFP_AFP

AFP_AFP:	Output file format AFPDS
AFP_PDFT:	Output file format PDF\TIFF (TIFF embedded in PDF)
AFP_FAX:	Output file format Topcall Fax
AFP_FXG3:	Output file format TIFF G3
AFP_IJP:	Output file format IJPDS
AFP_IPDS:	Output file format for IPDS-printers (via TCP\IP or channel card)
AFP_META:	Output file format Xerox Metacode
AFP_PCL:	Output file format PCL
AFP_PCL3:	Output file format PCL3 (for backwards compatibility only)
AFP_PCL4:	Output file format PCL4 (for backwards compatibility only)
AFP_PSCR:	Output file format Postscript
AFPTLECT:	Output file format for archiving

Using the specification AFPTLECT please see the *ISIS Papyrus Archive* documentation.

The attribute "Target mode (TargetMode)" of the class "MATERIAL" has the expression *ICD(TargetMode)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter TargetMode.

TargetType

```
TargetType = "{AFP_AFP, AFP_TIFF, AFP_GIF, AFP_PDF, AFP_PDFT}"
```

Defines the output document format for the archived files. This parameter is only relevant if **TargetMode=AFPTLECT**.

AFP_AFP	No conversion of the AFP documents
AFP_GIF	AFP will be converted to GIF
AFP_PDF	AFP will be converted to native PDF
AFP_PDFT	AFP will be converted to PDF\TIFF
AFP_TIFF	AFP will be converted to TIFF

For further details please see the *ISIS Papyrus Archive* documentation.

Default: AFP_AFP

The attribute "Target type (TargetType)" of the class "ARCHIVE AFP", which is derived from the base class "MATERIAL", has the expression *ICD(TargetType)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter TargetType.

PPMFForm

```
PPMFForm = "Filename without extension"
```

Defines the filename (without extension) of the FORMDEF (only used if the input AFPDS has no FORMDEF included).

Default: No default value.

The attribute "FormDef name (PPMFForm)" of the class "MATERIAL" has the expression *ICD(PPMFForm)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter PPMFForm.

PPMFLib

```
PPMFLib = "<Drive:>\Path\Filename" *on Unix /path/filename
```

Defines path and filename of the ISIS resource library profile where all AFP resource directories are defined.

Default: No default value.

The attribute "Resource library (PPMFLib)" of the class "AFPDS", which is derived from the base class "DATA", has the expression *ICD(PPMFLib)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter PPMFLib.

PPMFPPI

```
PPMFPPI = "{240|300|600}"
```

Specifies the resolution of the input AFP resources, either 240, 300 or 600 dpi.

Default: 240

The attribute "Resource resolution (PPMFPPI)" of the class "AFPDS", which is derived from the base class "DATA", has the expression *ICD(PPMFPPI)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter PPMFPPI.

AFPCodepage

```
AFPCodepage = "number"
```

Specifies the codepage to be used for interpreting the AFP index information from the input AFP file. It has to be set to the one which was used for the AFP creation. E.g. the US EBCDIC Codepage is 037, the German one 237 and the International one 500.

Default: 500

The attribute "Index codepage (AFPCodePage)" of the class "AFPDS", which is derived from the base class "DATA", has the expression *ICD(AFPCodepage)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter AFPCodepage.

CodepagePath

```
CodePagePath = "<Drive:>\Path\Filename" *on Unix: /path/filename
```

This parameter specifies where non standard codepage table files are searched for.

The codepage DLL **CDPDLL** used by all Papyrus products supports all commonly used codepages. If it is required to use a different one or to overwrite the internal codepage information an ISIS Papyrus binary codepage file can be placed into the specified path and the CDPDLL will read it from there.

A possible value could be the following:

```
CodepagePath="c:\isis\cpts"
```

Default: No default.

The attribute "Codepage path (CodePagePath)" of the class "PRINTER", which is derived from the base class "ISIS TOOL", can be used to specify the codepage path. The field "Default" of this attribute is set to the path "..\cpts".

DefaultFontLib

```
DefaultFontLib = "<Drive:>\Path\Filename" *on Unix: /path/filename
```

Defines path and filename of the ISIS resource library profile, in which the AFP default fonts are specified. Necessary only if the fonts to be used are not specified in the AFP.

Default: No default.

The attribute "Default font library (DefaultFontLib)" of the class "PRINTER", which is derived from the base class "ISIS TOOL", can be used to specify the path and filename of the ISIS resource library profile. The field "Default" of this attribute is set to "..\userisis\default.lbp".

DefaultDBCSFont

```
DefaultDBCSFont = "string"
```

Specifies the name of the default AFP DBCS (double byte character set) font, which will be used, if none is defined in the AFP.

The font name must be 1-6 characters long without a Xn prefix.

Default: No default.

The attribute "Default DBCS font (DefaultDBCSFont)" of the class "MATERIAL" has the expression *ICD(DefaultDBCSFont)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter DefaultDBCSFont.

DefaultSBCSFont

```
DefaultSBCSFont = "string"
```

Specifies the name of the default AFP SBCS (single byte character set) font which will be used, if none is defined in the APF.

The font name must be 1 to 6 characters long without the **Xn** prefix.

Default: No default.

The attribute "Default SBCS font (DefaultSBCSFont)" of the class "MATERIAL" has the expression *ICD(DefaultSBCSFont)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter DefaultSBCSFont.

PageFrom

```
PageFrom = "number"
```

Specifies the page number, at which Papyrus Server should start processing the print output.

Default: 1

The attribute "From page (PageFrom)" of the class "MATERIAL" has the expression *ICD(PageFrom)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter PageFrom.

PageTo

```
PageTo = "number"
```

Specifies the page number at which Papyrus Server should finish processing the print output.

If the value is set to -1, Papyrus Server will finish processing after the last page.

Default: -1

The attribute "To page (PageTo)" of the class "MATERIAL" has the expression *ICD(PageTo)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter PageTo.

CopyFrom

CopyFrom = "number"

Specifies the COPYGROUP number where the Papyrus Server should start processing the print output.

If the value is set to 0, Papyrus Server will start at the first COPYGROUP.

Range: 0-999999
Default: 0

The attribute "From CopyGroup (CopyFrom)" of the class "MATERIAL" has the expression *ICD(CopyFrom)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter CopyFrom.

CopyTo

```
CopyTo = "number"
```

Specifies the COPYGROUP number at which Papyrus Server should finish processing the print output.

If the value is set to 0, Papyrus Server will finish at the last COPYGROUP.

If this parameter is omitted, Papyrus Server will use the default value 0.

Range: 0-999999
Default: 0

The attribute "To CopyGroup (CopyTo)" of the class "MATERIAL" has the expression *ICD(CopyTo)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter CopyTo.

JobInfoText

```
JobInfoText = "string"
```

When the Adapter instantiates a Task template it automatically assigns the attribute "Description" of the new Task instance with the contents of the ICD parameter JobInfoText.

JobDisposition

```
JobDisposition = "{Release|Hold}"
```

Specifies if the job should be held or released.

Default: Release

The method parameter JobDisposition==R= is defined in the method "Send to printer" of the class "PJL", which is derived from the base class "ISIS TOOL". It is also possible to map the contents of the ICD parameter JobDisposition into an attribute of the Task or its children and pass it to this method parameter.

FileDisposition

```
FileDisposition = "{Delete|Keep}"
```

Specifies if the input file (or the control file) should be kept or deleted after processing.

Default: Keep.

Values:

Delete	The input file (or the control file) is deleted after successfully processing the print request.
Keep	The input file (or the control file) is kept after processing the print request.

The method parameter FileDisposition==K= is defined in the method "Send to printer" of the class "PJL", which is derived from the base class "ISIS TOOL". It is also possible to map the contents of the ICD parameter FileDisposition into an attribute of the Task or its children and pass it to this method parameter.

QueueManagerQueueName

```
QueueManagerQueueName = "string"
```

Specifies the name of the queue in which the Task should be processed.

The attribute "Processing queue (ProcessingQueue)" of the class ""PRINTER TASK", which is derived from the base class "TASK", has the match criterion *ICD($MATCH_RECURSIVE$,QueuemanagerQueueName)* defined in the field "Type property", thus the Adapter will instantiate the Task template if the contents of the ICD parameter QueueManagerQueueName equals the contents of this attribute.

AbendOnError

```
AbendOnError = "{Yes|No}"
```

Specifies whether processing should be abend in case of error or not.

Description: Whenever an error message is issued by an Application, this parameter will decide whether the application is supposed to continue working or to abend. However, if the application's RC is not interpreted as an error due to the setting of the ErrorLevel parameter, the AbendOnError parameter is ignored and the application will continue working. In case the application is abending, the module will still remain up&running.

Default: No

MaximumWarnings

```
MaximumWarnings = "0"
```

Specifies whether terminate the job with RC > 0 if the number of warnings exceeds the given amount.

Default: "0"

ReportMessageLevel

```
ReportMessageLevel = {I|W|E}
```

This parameter sets the level of messages that are to be put out by an ISIS module.

Values:

"I"	All message types are reported: I, W, E
"W"	Warning and Error messages are reported; Info messages are filtered away
"E"	only Error messages are reported; Info and Warning are filtered away

Default: "I"

The benefit of not sending too many messages, is evident in terms of conversion speed and LOG file size.

ErrorLevel

```
ErrorLevel = <Integer>
```

Sets the level at which a specific return code is to be interpreted as an error.

```
RC=4  ... warning
RC=8  ... error
RC=12 ... warning+error
RC=15 ... fatal

Recommendation: ErrorLevel=8

AbendOnError=Yes
```

The attribute "Error level (ErrorLevel)" of the class "ISIS TOOL" can be used to select the error level.

Samples:

If ErrorLevel=12 and an application returns RC=8, this will not be interpreted as error, thus the Material related to this step of a Task in Papyrus WebControl will not change to state "Error" but to "Final".

If ErrorLevel=8 and an application returns RC=4, this will not be interpreted as error, thus the Material related to this step of a Task in Papyrus WebControl will not change to state "Error" but to "Final".

If ErrorLevel=4 and an application returns RC=4, this will be interpreted as error, thus the Material related to this step of a Task in Papyrus WebControl will change to state "Error".

Special cases:

If ErrorLevel=0 and an application returns RC=0 this will not be interpreted as an error, even though the return code is as high as the ErrorLevel.

If RC >= 15, this will always be interpreted as error, regardless of the setting of ErrorLevel.

StartBanner

```
StartBanner = "<Drive:>\Path\Filename" *on Unix: /path/filename
```

Specifies an AFP file, which is used as the start page of the printed job, to help the printer operator to identify the printout of a job and to check statistics like number of pages and start / end time. Strings in this AFP that match the names of attributes in the Task used to print the job are replaced by the values of these attributes (e.g. StartDate, EndTime, etc.).

Default: There is no default.

EndBanner

```
EndBanner = "<Drive:>\Path\Filename" *on Unix: /path/filename
```

Specifies an AFP file, which is used as the end page of the printed job, to help the printer operator to identify the printout of a job and to check statistics like number of pages and start / end time. Strings in this AFP that match the names of attributes in the Task used to print the job are replaced by the values of these attributes (e.g. StartDate, EndTime, etc.).

Default: There is no default.

JobInfoText

```
JobInfoText = "string"
```

Specify any text description to the job.

Default: There is no default.

The attribute "Description (Description)" of the class "TASK" has the expression *ICD(JobInfoText)* defined in the field "Type property" to let the Adapter fill this attribute with the contents of the ICD parameter JobInfoText.

JournalPath

```
JournalPath = "<Drive:>\Path"
```

Specifies the path where the Papyrus Server should save the job information journal file ISISDISP.JNL.

The journalfile contains one line of job information, such as output type, duration, date, number of pages, and status about each print request which has been processed.

Default.working directory

Note: In Papyrus Objects, this information is written to the Journal object, which is attached to the Tool object.

MinFreeDiskSpace

```
MinFreeDiskSpace = "number"
```

Specifies if Papyrus Server should check for available free hard disk space.

The specified number is measured in megabytes.

If the specified free disk space is less than the value specified, Papyrus Server will stop until the required free disk space is available.

If the value is set at 0, Papyrus Server will not check for available free disk space.

Default::0

7.2.2 Parameter Summary for the Modules

For a detailed explanation of the module parameters see the module specific documentation.

7.3 Usage of the File Receiver in Papyrus WebControl

When processing an ICD file, the Adapter of the File Receiver must instantiate a Task template containing the appropriate Material. The class of the Material to be used can be determined via the parameter InputMode of the ICD file. The Adapter is able to instantiate such Task templates containing the appropriate Materials (AFP, ASCII, Metacode, and Line data) and put the Task instance into the correct Queue. The names of the templates are attributes of the Adapter. The name of the destination Queue is taken either from the QueueManagerQueueName parameter or from an Adapter attribute.

7.3.1 Mapping ICD parameters to attributes

1. The Adapter checks the templates of its General Container (Tasks and Materials along with Data objects) for match criteria in the "Type property" fields of their attributes. In one of the Task templates the Adapter finds the Match Criterion ICD($MATCH_RECURSIVE$,QueueManagerQueueName) in the "Type Property" field of the attribute "Destination". As the attribute contains the same value as the ICD parameter, the Adapter instantiates this Task.
2. The Adapter finds the match criterion ICD(PCL4TargetType,PCLTargetType,PCL3TargetType) in the "Type property" of the attribute "PCL type", which belongs to the Data object derived from the class "PCL". As the Adapter finds the matching ICD parameter "PCLTargetType" it stores the value in this attribute.
3. In the Material object derived from the class "PRINT AFP" the value of the attribute "PCL type" (Internal Name is PCLTargetType) is copied into PclTargetType (method Parameter 17) of the Material method "Print" via the expression PclTargetType=%PCLTargetType% (%PCLTargetType% refers to the "Internal name" of the Data attribute).
4. And finally the Material parameter is mapped into the respective Tool parameter of the Tool derived from the class "PCL - AFP_PCL".
5. The Tool parameter "PCLNrOfCopies" comes directly from a Tool attribute via %PCLNrOfCopies% which refers to the "Internal name" of the Tool attribute. Note that other Tasks (e.g. DocEXEC Task) can contain Materials where even Material parameters can come from their own attributes.

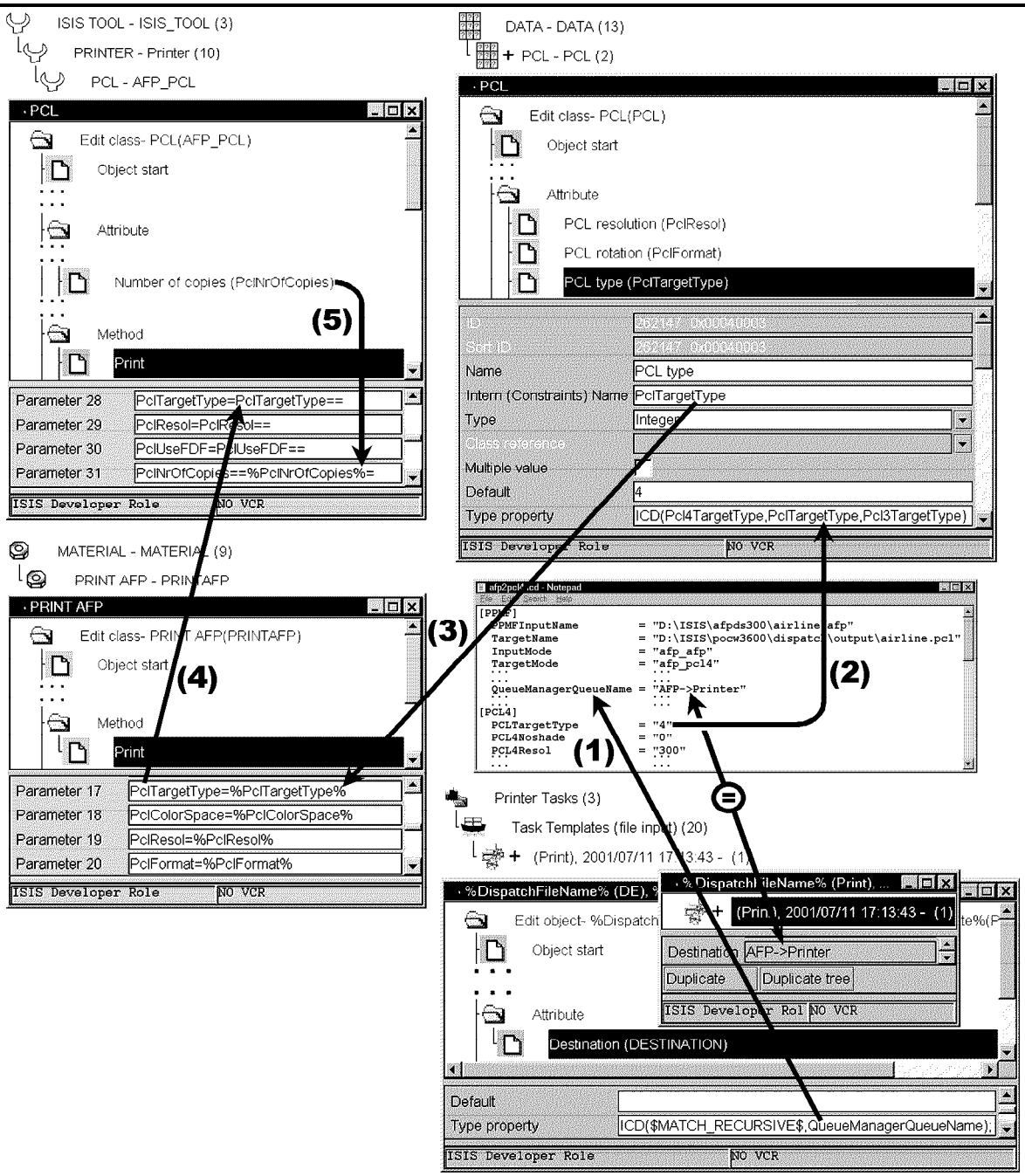


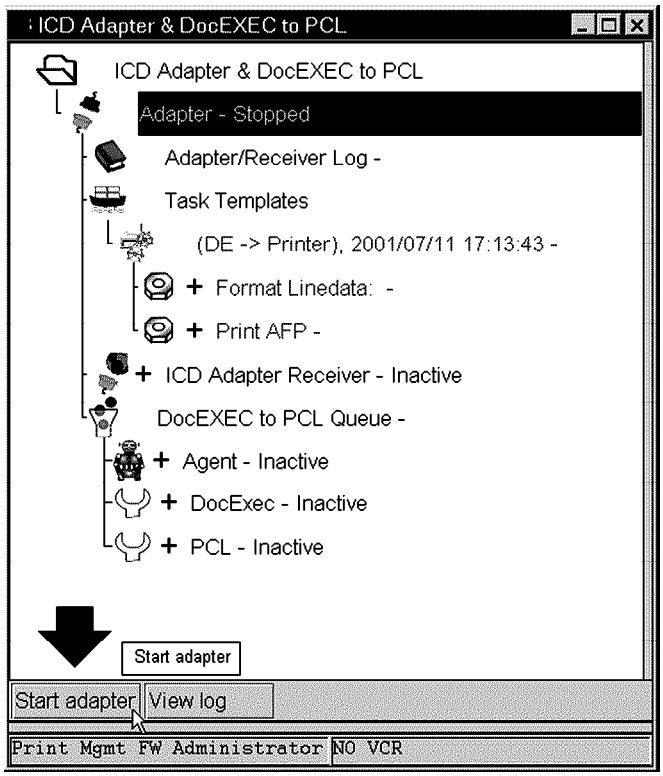
Figure 5: Mapping of the ICD parameters

7.3.2 Sample setup of a File Receiver

Instantiate the folder "ICD Adapter & DocEXEC to PCL" from the Library folder "Library / WebControl / Ready To Use Samples / ICD Adapter Samples" onto the local Node.

Check if the directory specified in attribute "Working directory" of the File Receiver is the one containing the dispatch.ini and verify that the directories specified therein (control, log, error, backup, output) do exist.

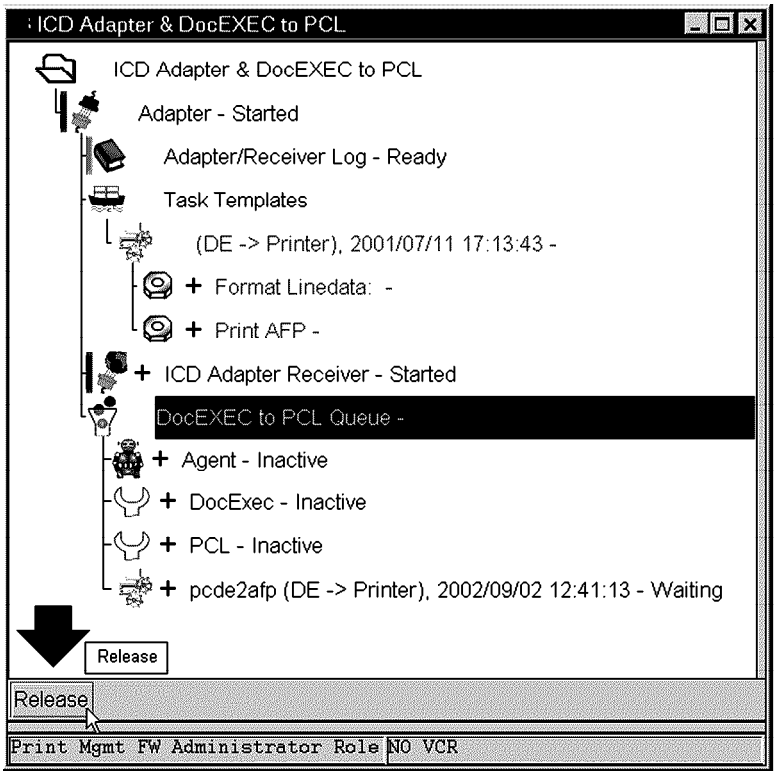
Start the Adapter by selecting it and clicking button "Start Adapter". Starting the ICD Adapter will cause the File Receiver to start instantly. All ICD files copied into the control directory will be processed.



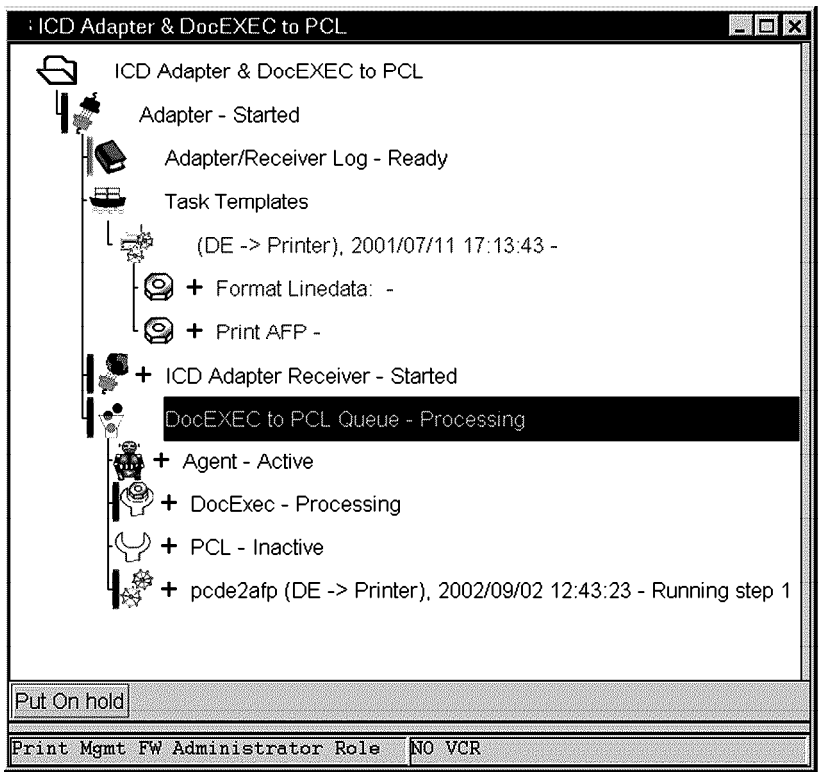
Copy ICD files from the ISIS sub directory "..\demo\webctrl\icd" into the ISIS sub directory "..\pocw3600\dispatch\control" (which one of the ICD files is intended for which Processing Queue can be found in the attribute "Description" of the respective Processing Queue).

The Adapter instantiates the appropriate Task template with the set parameters.

To process the instantiated Task the Queue has to be released. Right click the "Job Distribution Queue" and select "Release" from the context menu or click the respective button in the toolbar. This will automatically start the Agent which will move the Task into the appropriate Processing Queue.



The Task is processed and an output file "ASCIAirE.afp" is generated in the ISIS sub directory "..\demo\output\afp\".


ICD Adapter & DocEXEC to PCL
ICD Adapter & DocEXEC to PCL
Adapter - Started
Adapter/Receiver Log - Ready
Task Templates
(DE -> Printer), 2001/07/11 17:13:43 -
Format Linedata: -
Print AFP -
ICD Adapter Receiver - Started
DocEXEC to PCL Queue - Processing
Agent - Active
DocExec - Processing
PCL - Inactive
pcde2afp (DE -> Printer), 2002/09/02 12:43:23 - Running step 1
Put On hold
Print Mgmt FW Administrator Role
NO VCR

After our work is done put the Queue on hold by right clicking it and selecting "Put On hold" from the context menu or clicking the respective button in the toolbar.

Stop the Adapter by right clicking it and selecting "Stop Adapter" from the context menu or clicking the respective button from the context menu.

Check the Scheduler to be sure that no more waiting objects exist.

Finally close Papyrus Desktop (main menu "File | Exit" or click the close button right on the title bar), activate the window of the Papyrus Objects Kernel and press [CTRL] + [C] to properly shutdown the Papyrus Objects Kernel.

Note: If Papyrus Objects Kernel runs as a service there is no window associated to it and the shutdown works automatically.

8.0 PQL Based File Mapping

Instead of defining the file mapping in the dispatch.ini, it is also possible to use the template setup described in this chapter. It provides a PQL based file mapping solution for higher flexibility to administrate big amaounts of file mapping entries, but it is slower than the classic file mapping via the dispatch.ini. The dispatch.ini is still required but its parameters "FileStructur" and "FileMapping" are ignored and the File Mapping Entry objects (FM Entries) are used instead.

8.1 Configuration of the PQL Based File Mapping

File Mapping Entry objects (FM Entries) are used to define the file mapping for each expected type of file dropped into the control directory of the File Receiver (Dispatcher, ICD Receiver). File Mapping Group objects (FM Groups) are used to organize the FM Entries in groups. All FM Groups below the Adapter of the File Receiver are used to find the matching Task template for the input file dropped into the control directory.

8.1.1 Adapter Setup

To generate a new Adapter setup, duplicate the Adapter template of Library folder "Adapters/FileMapping/Adapter/Receiver with FileMapping". Check if the directory specified in attribute "Working directory" of the File Receiver template is the one containing the dispatch.ini and verify that the directories specified therein (control, log, error, backup, output) do exist.

Checking option "Trace" of the Adapter in this setup not only outputs a trace but some additional information about file mapping for each job (FileStructure, FileMapping, replaced ICD parameters, etc.).

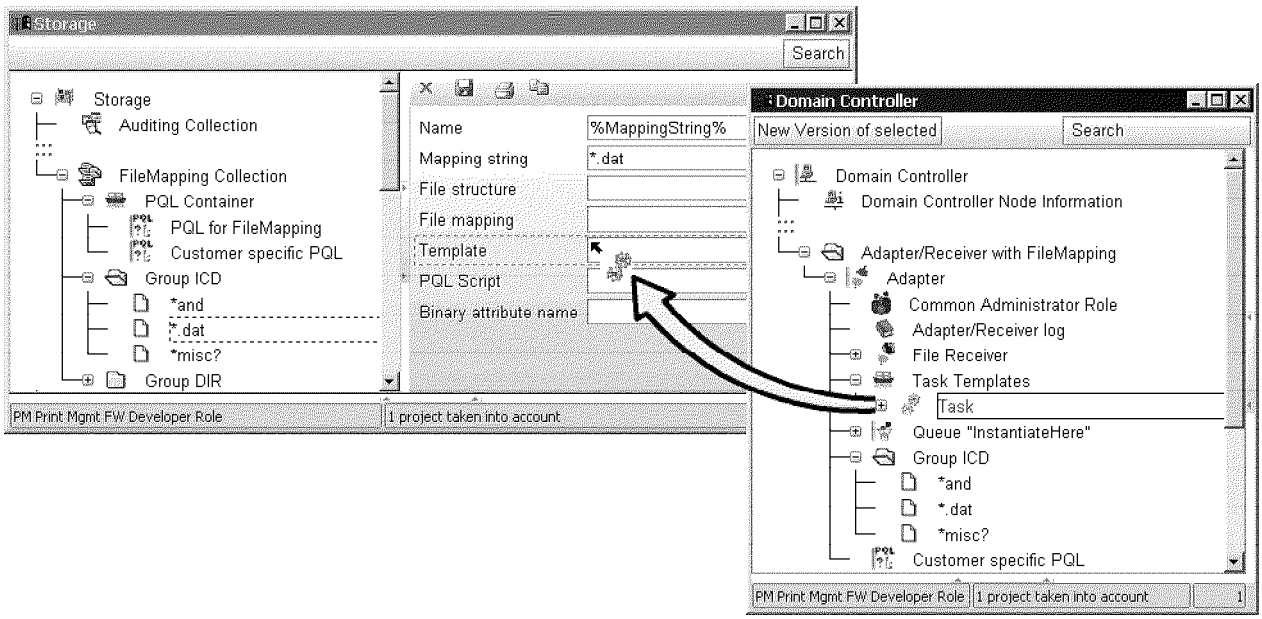
There is already one FM Group template containing three FM Entry templates and the PQL object "Customer specific PQL". An arbitrary number of FM Groups containing arbitrary numbers of FM Entry objects can be added to the Adapter, which uses them to find the appropriate Task template for the input file dropped into the control directory.

The Task template in the General Container below the Adapter is only an example. Usually for each FM Entry object a separate Task template has to be defined (referenced via attribute "Template" of the FM Entry object). The Queue below the Adapter is also only an example and can be extended by a whole workflow of Queues, Tools, and Agents including Rules.

8.1.2 FM Group and FM Entry objects

FM Group Attribute **"File structure"** defines the default file structure for its child FM Entry objects. Enter a mapping string in attribute **"Test file name"** and attribute **"Result"** returns all matching FM Entry objects via the PQL script of attribute **"Script"**. Attribute **"Group"** is an arbitrary caption for this FM Group.

FM Entry Define one FM Entry object for each expected file type and specify its search string in attribute **"Mapping string"** (e.g. *.dat to match all files with extension "dat"). While attribute **"File structure"** can be left empty to take over the file structure defined in the parent FM Group, the file mapping always has to be defined via attribute **"File mapping"**. To define which Task template shall be instantiated by the Adapter when this FM Entry object matches the input file, drop the Task template on attribute **"Template"** in the activated Data panel (requires two windows as depicted below). Attribute **"PQL script"** is reserved for an optional PQL script executed at the end (called by the PQL script of the PQL Rule). To automatically store the input file in a binary attribute of the Task and/or its child objects (usually a Data object), define its internal name in attribute **"Binary attribute name"**.



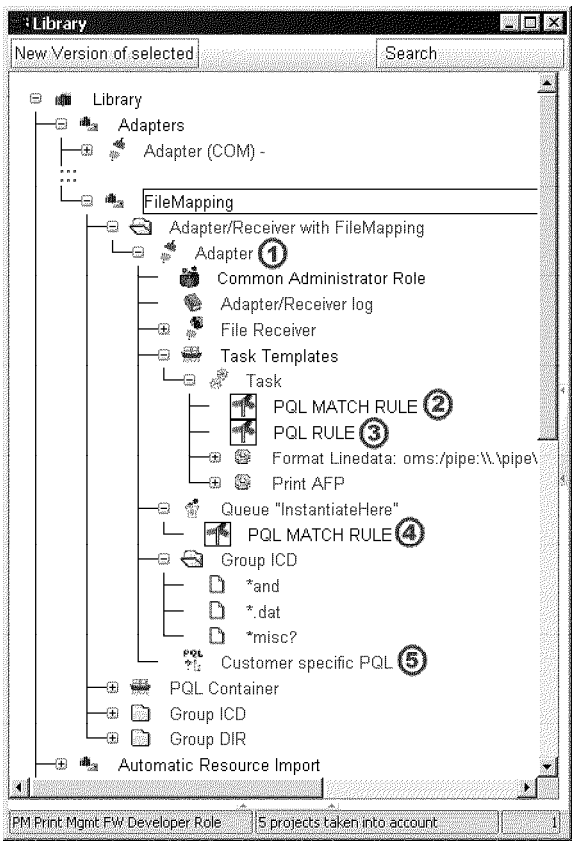
The FileMapping Collection in the Storage can be used to manage the FM Groups, their FM Entry objects, and PQL objects on one place by referencing them.

Important!

The mapping is performed in the sequence of FM Entry objects below the FM Group, i.e. put the FM Entries with the most specific mapping strings at the beginning (e.g. MyArchive01.txt) and the most general ones at the end (e.g. *).

8.1.3 PQL Scripts

The table below the screenshot gives a rough overview to the tasks performed by the PQL scripts, depicting the workflow of the file mapping. To adjust and extend these PQL scripts a deeper knowledge of the Papyrus Query Language (for details see chapter "PQL - Papyrus Query Language") is required.



(1) As the PQL script defined in the Source Alias is always executed when the Receiver passes data to its Adapter (in this case when dropping a file into the control directory) this is the place to find the appropriate Task template and by this perform the file mapping based on FM Entry objects.

PQL sets for the aliases are automatically generated and the variables Source and Adapter only have to be declared (dim Source,Adapter;) to access these PQL sets.

Variables are assigned with the values passed by the Receiver via $name and $value (e.g. @filename=Source\$allchildren(*:.**$name**=="DispatchFileName").**$value**;).

The name of the file dropped into the control directory (combination of parameters DispatchFileName and DispatchFileExt, automatically generated by the Receiver) is compared with attribute "Mapping string (MappingString)" of all FM Entry objects in the FM Groups below the Adapter, to find the appropriate Task template. The Object ID of the Task template, defined in attribute "Template (Template)" of the found FM Entry object, is stored in variable @template.

If attribute "File mapping (FileMapping)" of the FM Entry object is not empty, variable @mapping is assigned with it and extended with the data passed by the Receiver. Otherwise, the values are directly taken from the input filename instead.

If attribute "File structure (FileStructure)" of the FM Entry object is not empty, variable @structure is assigned with it and extended with the ICD parameters corresponding to the data passed by the Receiver. Otherwise attribute "File structure (FileStructure)" of the parent FM Group object is taken as default.

The "Customer specific PQL" **(5)** below the Adapter is called and the variables @filename and Adapter.trace are passed to it, which can be accessed there as $param4 and $param5.

Finally, if the Adapter trace is switched on, the values of some variables are output in the kernel window.

(2) Variable @template, assigned with the Object ID of the Task template in the PQL script of the Source Alias **(1)**, can be accessed by declaring it again in this PQL script.

Variable @template is the equivalent to the ICD file at the first position of the file structure defined in the dispatch.ini using the classic approach. Thus instead of using an ICD file to populate the attributes of the Task instance generated by the Adapter, the attributes are already assigned with the required values in the Task template and only the attributes defined in the FM Entry object are overwritten in the Task instance.

If the Object ID in variable @template, which is the Task template defined in the FM Entry object, is identical with the own Object ID, the parent Task template gets instantiated by the Adapter with all its children ($MATCH_RECURSIVE$) and the PQL script in the PQL Rule **(3)** is executed.

(3) The variables @structure and @mapping, generated in the PQL script of the Source Alias **(1)**, can be accessed by declaring them again in this PQL script. They are used to populate the matching attributes (internal name = name of ICD parameter) of the new Task instance (available by declaring variable instance) and all its child objects (variable @objects contains all child objects).

The placeholder **thisfile** in the mapping string is replaced by path and filename of the input file. As the dispatch filename might contain hashes, it is put at the end of the file mapping string, which uses hashes as delimiter.

If the trace is switched on, the PQL script also informs which parameters have been replaced by which values in the kernel window.

If a binary attribute has been specified in attribute "Binary attribute name (BinaryAttributeName)" of the FM Entry object (stored in variable @entry of PQL script in Source Alias), the input file is stored in the matching binary attributes of the Task instance and all its child objects (variable @objects contains all child objects).

Finally the optional PQL script defined in the FM Entry object (strored by the PQL Script of the Source Alias **(1)** in variable @pqlscript) is executed.

(4) @result, generated in the PQL script of the Source Alias **(1)**, can be accessed by declaring it again in this PQL script.

If attribute "Move target (MoveTarget)" of the Queue is identical with the string stored in @result.ResultString, the Adapter instantiates the Task template into this Queue ($MATCH_DESTINATION$).

(5) Called by the PQL script of the Source Alias **(1)**, returning the string "InstantiateHere" used to find the Queue where the Adapter shall instantiate the Task template. Can be extended to perform additional organization specific tasks (e.g. return string based on parameter received from calling PQL script).

8.2 Usage of the PQL Based File Mapping

Add an instance of the Adapter template on the desired Production Node, select the new Adapter, and click "Start Adapter".

Drop a file into the control directory of the File Receiver and the Adapter will instantiate the matching Task template using the FM Entry objects in its FM Groups to find it.

9.0 XML Receiver

The XML Receiver is a file dispatcher with XML capabilities, it checks the defined directory for incoming files that match a certain filter, processes them, and passes them to the Adapter.

Features of the XML Receiver

- The document has to be enclosed by tags (container) which are definable.
- The Part can be extracted as a BLOB.
- One container is required per document.

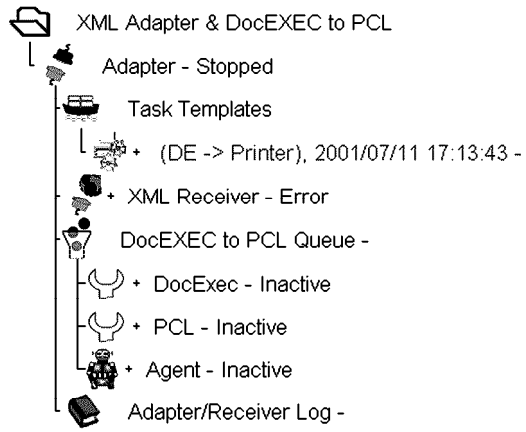

XML Adapter & DocEXEC to PCL
Adapter - Stopped
Task Templates
+ (DE -> Printer), 2001/07/11 17:13:43 -
+ XML Receiver - Error
DocEXEC to PCL Queue -
+ DocExec - Inactive
+ PCL - Inactive
+ Agent - Inactive
Adapter/Receiver Log -

Figure 6: Typical setup for a XML Receiver

9.1 Components of the XML Receiver

9.1.1 Attributes of the class "XML Receiver"

Program (Program)	reference to the Program object to automatically start the module for the currently running operating system.
Parameter dump (ParameterDump)	If checked, the start parameters are dumped.
Trigger directory (reqdir)	Specifies the directory to monitor for incoming files (monitoring step).
File extension (reqext)	All files containing the specified string in their name (e.g. the extension .xml) are processed (filtering step).
Processing directory (tempdir)	Specifies the processing directory (processing step).
Document tag (blockname)	Specifies the tag enclosing the whole xml document (e.g. <DOCUMENT>). To predefine a docment tag, enter its name in the field "Default" of this attribute in the respective template (e.g. Default = DOCUMENT).
Container tag (contain)	Specifies the tag used for data that should not be converted to name / value pairs by the XML Receiver in mode 1. The contents is not being analyzed but passed to the Adapter directly "as is", which can be of type String or Binary (e.g. BLOB, channel code or XML file required by Papyrus DocEXEC as line data, etc.). If no container tag has been defined, the default container tag <container> will be used.
Triggered file (filename)	This attribute is used to define an identifier for the name / value pair automatically generated by the XML Receiver to store path and filename of the XML file dropped into the trigger directory (only avaliable in mode 1). An additional attribute in the Task template or one of its child objects can be used to retrieve path and filename of the XML file via a match criterion (e.g. via the match criterion XML(TriggerFile) defined in attribute "Dispatch filename (DispatchFileName)" of the Task templates below "Library / WebControl / Ready To Use Samples / XML Adapter Samples / XML Adapter & Distribution Queue / Adapter - %Objstate%/Task templates" path and filename of the XML file can be retrieved as the string "TriggerFile" has been defined in attribute "Triggered file (filename)" of the XML Receiver).

Adapter Mode (adaptermode)

The following options are used to specify how the XML Receiver processes the incoming XML data to pass it to the Adapter (if no option is selected, mode 1 is taken by default).

mode1

In this mode the **XML Receiver converts the tags into name / value pairs** and ignores all tags that are used as containers for other tags (e.g. tags <TASK>, <INPUTPART>, and <OUTPUTPART> in the example below). In this mode only match criteria in the field "Type property" of the attributes defined in the template used for instantiation and its child objects can be used by the Adapter to process the XML data. Therefore the templates used for instantiation and their child objects may not contain Rules, otherwise the match criteria are ignored by the Adapter. Data between container tags is not interpreted by the XML Receiver but passed to the Adapter as-is (see attribute "Container Tag (contain)" above).

In this mode the files, specified in tags with the keyword storage="file", can be imported into binary attributes via match criteria, whereas each tag used for the file import must be enclosed by <dataset> tags like in the following example.

```
<XML>     <collection>         <dataset>            <Content
storage="file">c:\isis\data\input.afp</Content>
...other  XML  content...        </dataset>     </collection>
...other XML content... </XML>
```

There may be multiple occurrences of <dataset> containers - the Adapter will iterate through these and instantiate a corresponding template substructure for each of them, provided that the template substructure contains a matching attribute with the TypeProperty "XML($MATCH_ITERATION$,...)".

Example:
In this example the following XML file is dropped into the trigger directory. <SAMPLEDIST> is defined as document tag and no container tag is defined.

```
<XML>
  <SAMPLEDIST>
    <TASK>
          <DESCRIPTION>This is a AFP to PCL sample job!</DESCRIPTION>

<PROCESSINGQUEUE>PCLQueue</PROCESSINGQUEUE>

<COMPLETEDQUEUE>CompletedQueue</COMPLETEDQUE
UE>
      <ERRORQUEUE>ErrorQueue</ERRORQUEUE>
    </TASK>
    <INPUTPART>

<AFPDS_NAME>..\afpds300\pnc5bw.afp</AFPDS_NAME>
    </INPUTPART>
    <OUTPUTPART>

<TARGETNAME>..\demo\output\pcl5\pnc5bw.pcl</TARGETN
AME>
    </OUTPUTPART>
  </SAMPLEDIST>
</XML>
```

The XML Receiver generates the following name / value pairs with the XML file above:

```
TriggerFile="C:\ISIS\PODev021\POCW3610\.\xml\process\SampleDi
stPCL.xml"
   DESCRIPTION="This is a AFP to PCL sample job!"
   PROCESSINGQUEUE="PCLQueue"
   COMPLETEDQUEUE="CompletedQueue"
   ERRORQUEUE="ErrorQueue"
   AFPDS_NAME="..\afpds300\pnc5bw.afp"
   TARGETNAME="..\demo\output\pcl5\pnc5bw.pcl"
```

mode2
In this mode the **XML Receiver passes the XML data as-is to the Adapter** to process arbitrary structured XML files. If the Adapter finds Rules in the templates used for instantiation and its child objects, it uses the PQL statements defined therein to process the tags, otherwise it uses the match criteria in the field "Type property" of the attributes defined in the template used for instantiation and its child objects.

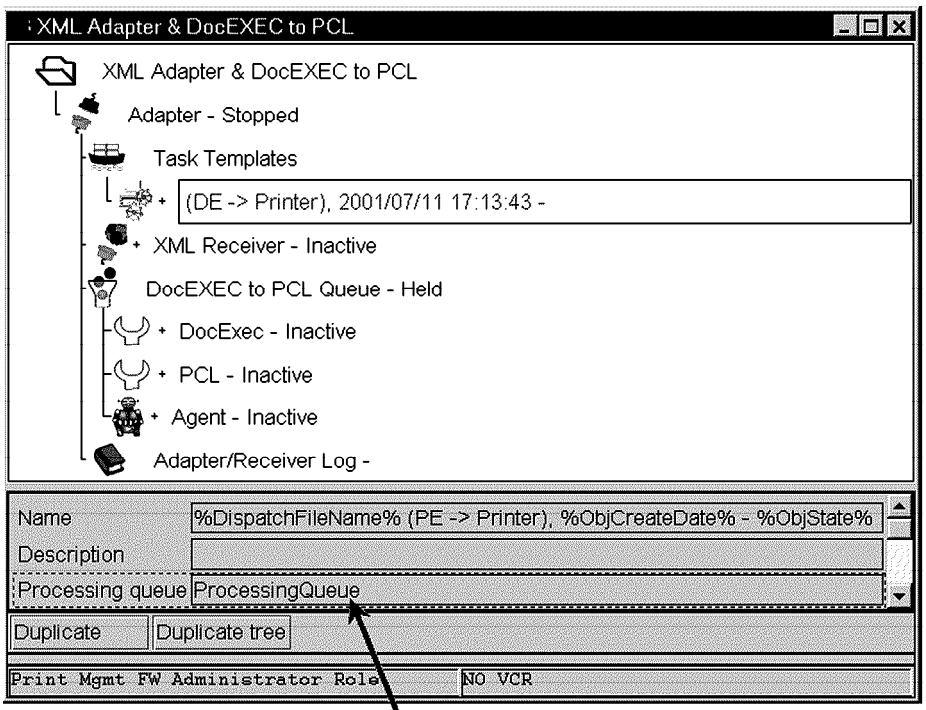
Container flag (containflag)

If checked, the container tag is included in the datastream as begin and end tag, otherwise it is omitted.

Error directory (errordir)	In case of a parser error the file gets moved into this directory.
Error file (errorfile)	Specify path and filename of the XML file that should be used instead of a XML file that produced a parser error.
File Name Flag (filenameflag)	If checked, path and filename of the XML file will be stored in the attribute specified via attribute "Triggered file (filename)", otherwise only the filename will be stored.
Confirmation (confirmflag)	If checked, the Adapter sends a confirmation to its XML Receiver as soon as it could successfully instantiate a template.
Confirmation timeout (confirmtimeout)	Value in seconds to wait for the confirmation sent by the Adapter. If attribute "Confirmation" is checked and the timeout runs out, a message is issued on Papyrus Desktop.
Delete processed File (delprocfileflag)	If this attribute and attribute "Confirmation" are checked, the XML file is deleted from the processing directory as soon as the Adapter sends a confirmation to its XML Receiver, that it could successfully instantiate a template.
Log directory (readydir)	Processed files are moved into this directory if processing was successful.
Use xercesflag (xercesflag)	If this option is checked, the XML Receiver makes use of the Xerces DLL (like other XML processing products by ISIS like Papyrus DocEXEC and PQL), which is a more rigid XML parser than the one used by default (e.g. to use XML data within an XML tag, it is mandatory to use a CDATA section: "<![CDATA[" and ends with "]]>").

9.1.2 Instantiation of the appropriate Task template via Match Criteria

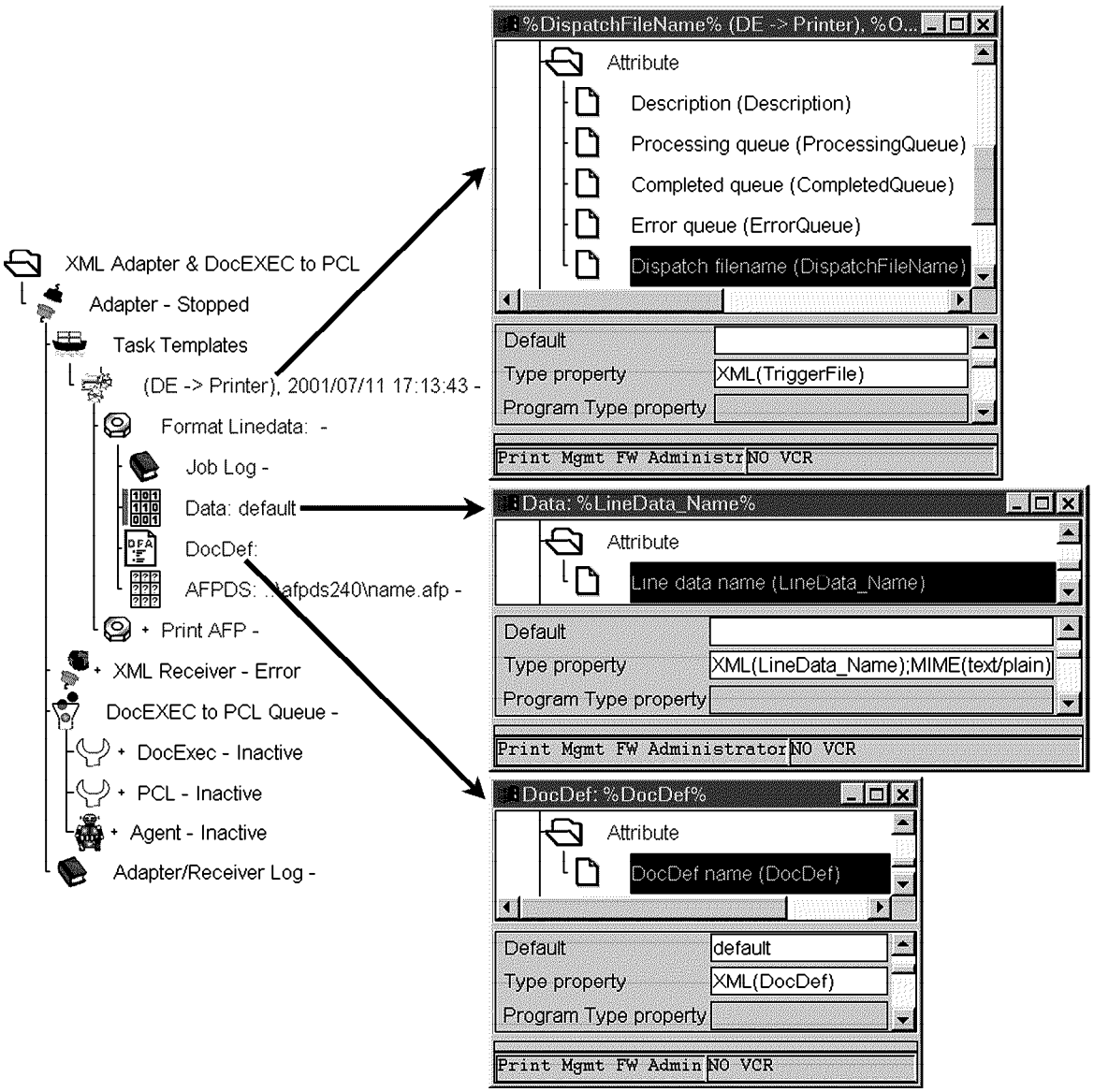
To enable the Adapter to decide which Task template should be instantiated depending on a value of the XML file a Match Criterion with one of the keywords ($MATCH_RECURSIVE$, $MATCH_LOCAL$ or $MATCH_ITERATION$) is required in the field "Type Property" of one of the Task attributes. As the attribute "Processing queue" of the Task template contains the Type Property **"XML($MATCH_RECURSIVE$,ProcessingQueue)"** the Adapter will check if this attribute value equals the value of the tag <PROCESSINGQUEUE> of the XML file. As both, attribute and tag, contain the same value, the Adapter instantiates the Task template into the "Processing Queue".




```
<XML>
  <SAMPLEAIR>
    <TASK>
      <DESCRIPTION>This is a DE->PCL sample job!</DESCRIPTION>
      <PROCESSINGQUEUE>ProcessingQueue</PROCESSINGQUEUE>
      <COMPLETEDQUEUE>CompletedQueue</COMPLETEDQUEUE>
      <ERRORQUEUE>ErrorQueue</ERRORQUEUE>
    </TASK>
    <LINEDATA_NAME>..\data\asciaire.asc</LINEDATA_NAME>
    <DOCDEF>..\docdef\ASCIAirE</DOCDEF>
    <AFPDS_NAME>..\demo\output\afp\XMLAirE.afp</AFPDS_NAME>
  </SAMPLEAIR>
</XML>
```

9.1.3 Mapping XML data into attributes

The parameter "TriggerFile" is generated automatically by the Adapter and contains the name of the XML file which was dropped into the trigger directory. As the attribute "Dispatch filename" of the Task contains the "Type Property" **XML(TriggerFile)**, the Adapter fills this attribute with the value of the automatically generated parameter "TriggerFile". All other attributes containing Match Criteria in their "Type Properties" are filled with values from the respective XML tags.



9.1.4 Predefined Entities

If the text between tags should contain special characters, like the greater than sign in the following example, use the respective entity values instead:

`<LIMIT>Order > 1000</LIMIT>` should read: `<LIMIT>Order > 1000</LIMIT>`

Specify an XML header and specify either "utf-8" or "utf-16" for the encoding, otherwise the XML Receiver ignores the entity values:

```
<?xml version="1.0" encoding="utf-16"?>
```

The XML Receiver translates the entity values back, thus the attribute "LimitInfo" with the Match Criterion *XML(Limit)* in the field "Type Property" would be filled with the string "Order > 1000".

The following table shows all predefined entities supported by the XML Receiver:

Character	Name	Entity value
<	less than sign	<
>	greater than sign	>
&	ampersand	&
'	single quote or apostrophe	'
"	double quote	"

Note: Other entity values can be used as well but will not be translated back by the XML Receiver. Special characters can also be specified by the number according to the HTML/XML standard (e.g. © instead of © for the copyright symbol).

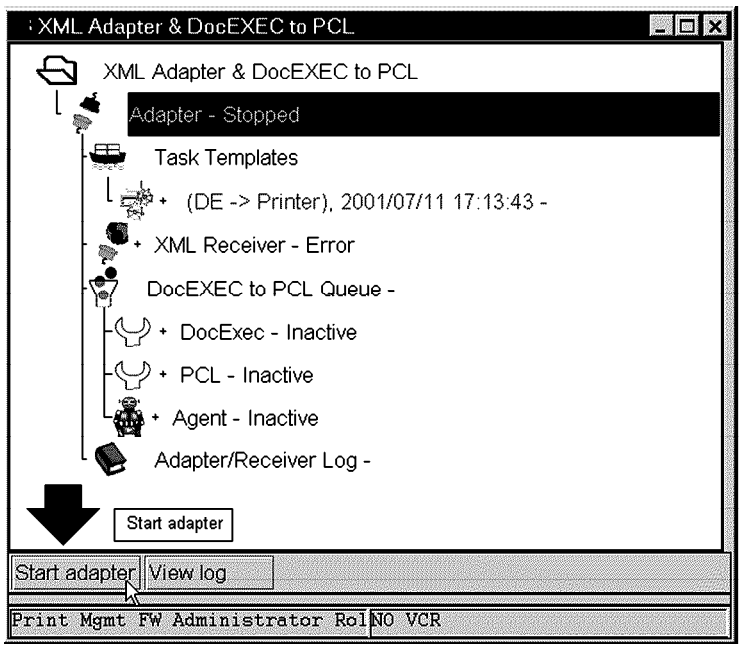
9.2 Usage of the XML Receiver

Log in as Print Administrator (default password = akira).

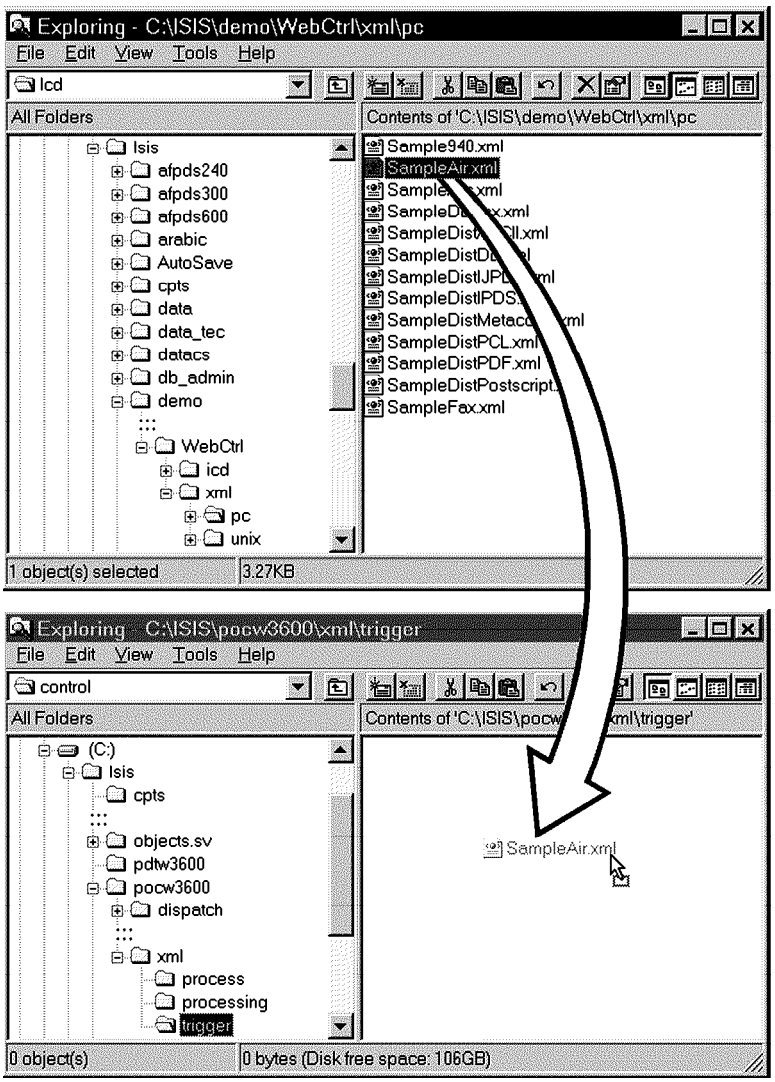
Instantiate the folder "XML Adapter & DocEXEC to PCL" from the Library folder "Library / WebControl / Ready To Use Samples / XML Adapter Samples" on the local Node.

Select the XML Receiver and modify the paths for the trigger directory and the processing directory if required.

Start the Adapter by selecting it and clicking the button "Start adapter" in the toolbar, this automatically starts the Receiver which monitors the trigger directory for incoming XML files.



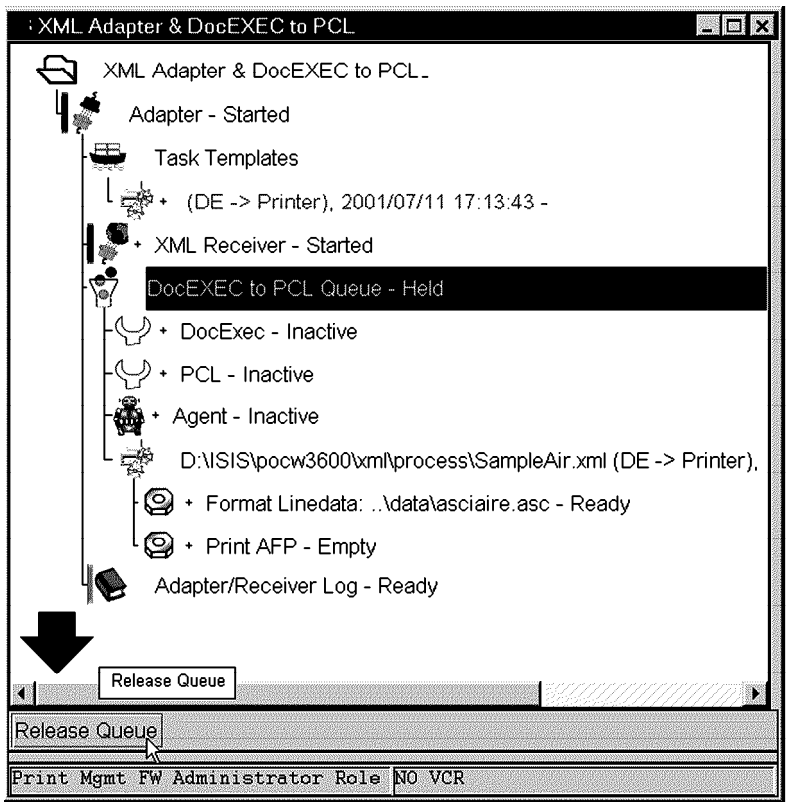
Copy an XML file (e.g. SampleAir.xml) into the trigger directory.



The Adapter instantiates the matching Task template into the "Processing Queue".

The Tool "PCL" is by default set that way no print out is generated and the printer port is set to LPT1, thus enter the name of the printer in the field "Name of queue or port" and check the checkbox "Print file" to output the generated PCL on the specified printer.

Release the Queue ("Release Queue" button) to start the document generation. The finished document is output on the printer.



10.0 Fax Receiver / Sender

The Fax Receiver / Sender is used to store incoming faxes as binary objects (TIFF files) and to send faxes. Therefore the Fax Receiver / Sender can be added either as part of a Correspondence Framework to fax the generated documents or as part of a Capture Framework to retrieve fax messages. To send each document of an AFP file as fax message, the FAXG3 Tool is used in advance to generate TIFF/G3 files which are stored as binary objects (in instances of the class "FAX (Binary)").

The receiver part or the Fax Receiver / Sender, which constantly listens to the serial COM port for incoming data (multithreaded DLL), reads the incoming fax pages and stores the data in temporary G3 files. As soon as the fax has been completely received the G3 files (one for each page) are merged into TIFF files and stored as binary objects in instances of the class "FAX (Binary)" as Data objects attached to the Material object.

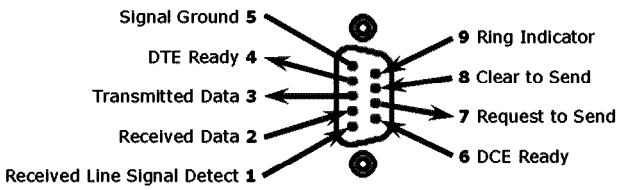
By selecting the Adapter and clicking the button "Start adapter" the communication between modem and computer is initialized. During this period of time it is not possible to send or receive fax messages, therefore the Fax Receiver / Sender is in state "Busy". As soon as the initializing process has finished, the Fax Receiver / Sender changes its state to "Ready" and is ready to receive fax messages. The method "Stop adapter" puts the Adapter and its related Fax Receiver / Sender on hold.

Main features of the Fax Receiver / Sender:

- Receiving of multi-page faxes at a bitrate of 2400 - 14400 bps, according to the installed fax modem, the transmitter modem and the signal quality of the telephone line. The received fax documents are automatically instantiated as binary objects containing multi-page TIFF G3 images.
- Sending of multi-page fax documents at a bitrate of 2400 - 14400 bps, according to the installed fax modem, the transmitter modem, and the signal quality of the telephone line.
- Optional user defined limitation of the maximum transmission speed, to reduce failure rate in case of poor S/N ratio on the telephone line.
- Support for Nomal and Fine resolution.
- Optional header line with user defined ID string (ASCII characters), time stamp and page count for each transmitted fax page.

Transmitted fax documents must comply with the Group 3 standard. Hardware requirements include a fax modem of Fax class 2 or 2.0 (refer to the table "Supported modems"), a Windwos 32-bit platform running a Papyrus Objects, and an analogue telephone line.

For the serial communication an UART (Universal Asynchronous Receiver Transmitter) compatible with 16550 has to be used. The following figure depicts a DB-9 male connector layout (modem to PC).

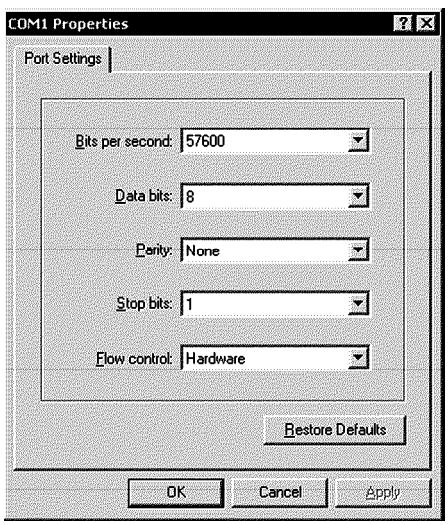


D	Description	Name	Pin
I	Data Carrier Detect	DCD	1
I	Receive Data	RxD	2
O	Transmit Data	TxD	3
O	Data Terminal Ready	DTR	4
-	Signal Ground	GND	5
I	Data Set Ready	DSR	6
O	Request To Send	RTS	7
I	Clear To Send	CTS	8
I	Ring Indicator	RI	9
-	Chassis Ground	GND	Ch

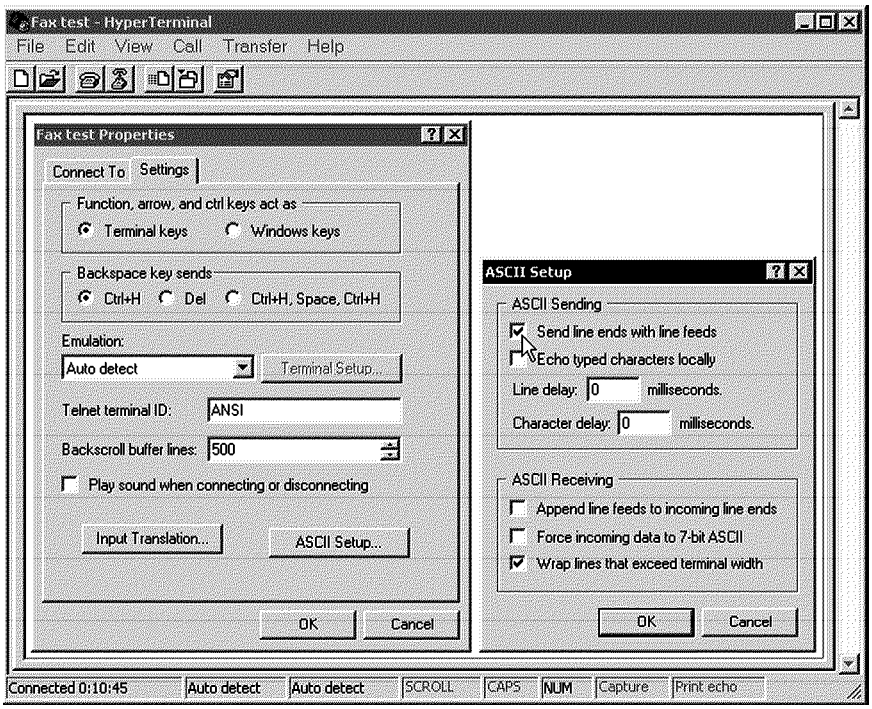
10.1 Hardware installation and configuration

10.1.1 Determine the COM port

If the COM port the fax modem is connected to is unknown, it is possible to identify it with a third-party serial communication application like "Hyper Terminal". Connect "Hyper Terminal" to the first port available (e.g. COM1) and use the same communication settings as intended for the Fax Receiver / Sender (e.g. 57600,8,N,1,HW flow control).



In the properties dialog click the button "ASCII Setup..." and select the checkbox "Send line ends with line feeds", otherwise the modem will not answer to Hayes commands (string followed by carriage return / line feed characters).



Type in the Hayes command "AT" in the main window and press [Enter]. If the modem is properly connected and powered on it should reply with "OK". If no response is output in the main window, the port number is wrong or the modem is not properly connected.

If the modem does not respond on any COM port although it is properly plugged in it might be jammed. To unjam the modem, try the following steps, in the given sequence:

- Issue the Hayes command "ATZ" and press [Enter], which resets the current modem profile to factory defaults.
- Issue three plus characters **("+++")** in fast sequence and press [Enter], which should unlock the modem from data mode and return to command mode.
- Power off the modem, wait for at least 30 seconds, to provoke a cold reset on the modem firmware which restores the modem to factory defaults.

The serial cable provided with the modem should work fine with any PC, but in case of doubts, check that the cable is non-inverting (rx/tx pin inversion is already implemented within the PC's UART).

10.1.2 Determine the fax class of the modem

To determine the fax class of the modem, type in the Hayes command "AT+FCLASS=2" and press [Enter]. If the modem responds with "OK" it is of fax class 2, if it responds with "ERROR", enter the Hayes command "AT+FCLASS=2.0" and [Enter]. If the modem responds with "OK" it is of fax class 2.0. If the modem responds to both Hayes commands with "ERROR", it is neither of fax class 2 nor 2.0 and cannot be used in combination with the Fax Receiver / Sender.

10.1.3 Identify the model

Once the COM port and the fax class of the modem is determined, it may be required to know its model and firmware revision level. Three Hayes commands can be issued via Hyper Terminal to get this kind of information.

Enter the following Hayes commands if the modem is of fax class 2:

```
AT+FMFR?[Enter] - returns the manufacturer name.

AT+FMDL?[Enter] - returns the model name.

AT+FREV?[Enter] - returns the firmware revision.
```

Enter the following Hayes commands if the modem is of fax class 2.0:

```
AT+FMI?[Enter] returns the manufacturer name.

AT+FMM?[Enter] returns the model name.

AT+FMR?[Enter] Returns the firmware revision.
```

10.1.4 Supported modems

Due to the possible differences in the fax class 2 and 2.0 implementation on different modems, it is recommended to use one of the models list below. Other models may also work without the requirement to adjust their setup, but a test should be performed in advance to verify full compatibility. It might happen that the receiver part and / or sender part of the Fax Receiver / Sender is not operational when using a model not listed below. For the modem models below, the setup is already defined in Papyrus Objects. For any other modem, the PO setup shall be created according to the modem characteristics. The fields "Manufacturer", "Model ID", and "Revision ID" are the strings returned by the modem when using the Hayes commands described in the section above.

Modem Name	**UsRobotics 56k Faxmodem**
Manufacturer	U.S. Robotics 56K FAX
Model ID	-
Revision ID	-
Fax	class 2.0
Initialization string	AT&F&H3&I2&R2S7=90L1M1X3
Xon expected	No
Rx Bit order reversed	Yes

Modem name	**USRobotics Sportster 33.6 fax modem**
Manufacturer	U.S. Robotics Sportster V.34+ FAX
Model ID	-
Revision ID	-
Fax	class 2.0
Initialization string	AT&F&H3&I2&R2S7=90L1M1X3
Xon expected	Yes
Rx Bit order reversed	Yes

Modem name	**ELSA Microlink 56k**
Manufacturer	ELSA AG, Aachen (Germany)
Model ID	MicroLink 56k Internet c
Revision ID	Version 1.85 / 28.04.2000
Fax	class 2.0
Initialization string	AT&F&C1&D2&K3S0=0S7=60L2M1X3
Xon expected	Yes
Rx Bit order reversed	Yes

Modem name	**Zyxel Comet 3356**
Manufacturer	ROCKWELL
Model ID	AC/K56
Revision ID	V2.200-V90_2M_DLS
Fax	class 2
Initialization string	AT&F&C1&D2S0=0S7=60L0M0X3&K4
Xon expected	Yes
Rx Bit order reversed	No

Modem name	**Magik 56k**
Manufacturer	CONEXANT
Model ID	V90
Revision ID	P2107-V90
Fax	class 2
Initialization string	AT&F&C1&D2S0=0S7=60L2M1X3
Xon expected	Yes
Rx Bit order reversed	Yes

10.1.5 Flow control settings

An incorrect modem initialization or non-matching serial line settings can cause random errors during data transmission, which can be quite hard to diagnose. Flow control in serial communications provides a mechanism for suspending communications between the host (PC) and the attached device (local modem) while one of the two is busy or for some reason cannot do any communication. There are traditionally the two types hardware and software flow control. The attribute "Flow Control" of the Fax Receiver / Sender is used to configure the COM port to use independently one or the other flow control or both types of flow control at the same time via one of the following options:

Disabled	Hardware and Software flow control are disabled on the PC side: Signal CTS is not sensed by the PC during output control and signal RTS,DTR are kept low; characters XON and XOFF are treated as data bytes, not as control characters so they are passed over to the application software (POCxxFAX).
Enabled	Same as option "Disabled', but signals RTS and DTR are always kept High (ON) during connection. Modems are normally sensitive to these signals, so they won't start communication if the signals are not set. Note: Fax Receiver / Sender always checks for Software flow control characters during FAX send phase C, so if the modem is properly configured, Software flow control should work correctly despite of this setting.
HW Handshake	Hardware flow control is activated: Signal CTS is sensed by the PC during data transmission (FAX send) and signal RTS is driven according to the UART's buffer needs, during data receive (FAX receive). Software flow control is not activated in the UART, so POCxxFAX receives and handles XON/XOFF characters if the modem is configured to send them.
SW Handshake	Hardware flow control is not activated (same as "Enabled"). Software control characters XON, XOFF are handled by the UART so the application software does not receive them.
HW+SW Handshake	Hardware signals are handled in the same manner as with option "HW Handshake". Software control characters XON, XOFF are handled by the UART in the same way as "SW Handshake" setting does.

On the other side, the modem must be configured accordingly via the initialization string, defined in the attribute "Init Strint (InitString)" of an instance of the class "FAX MODEM" attached to the Fax Receiver / Sender.

It is a good practice to keep the serial transmission speed between PC and the modem significantly higher than the maximum data speed on the telephone line. In fact, the PC must always feed the local modem fast enough when data is being sent to the remove machine (during fax send) to avoid the "data buffer underflow" error. During fax receive, data must be read fast enough by the PC to avoid buffer overflow inside the attached modem. As the maximum line speed for FAX communications is 14400bps (attribute "MaxBitRate" of the class "FAX RECEIVER"), the local line speed should be kept above 19200bps (attribute "Bit rate (BitRate)" inherited in class "FAX RECEIVER" by the base class "SERIAL RECEIVER"). Typically 57600bps is used, but 115200bps is not infrequent.

By default, most modems should have both software and hardware flow control enabled. This setting should remain unchanged, except in few special cases as detailed here below.

Hardware flow control requires (as its own name indicates) some hardware on the communication cable, in addition to three basic lines TxD, RxD and GND. Two dedicated lines are used to transmit the signals "Clear To Send (CTS)" and "Request To Send (RTS)", which serve the following purpose:

- When the modem receives data from the PC (for instance, during FAX Send phase) and requires to temporarily suspend data flow, it lowers signal CTS.
- When the PC receives data from the modem (for instance, during FAX Receive phase) and needs to temporarily suspend data flow, it lowers signal RTS.

10.1.5.1 Hardware flow control

Hardware flow control must also be activated on the modem through the initialization string, otherwise the modem doesn't know if the hardware signals are in use. Unfortunately, modem manufacturers use different commands for this purpose.

For instance, with U.S. Robotics 56k fax-modems signal CTS is enabled by command "&H1" during data transmit phase (fax send) and "&R2" senses the level of signal RTS during data receive phase (fax receive). To summarize, the following initialization string enables input and output HW flow control on U.S. Robotics 56k fax-modems:

```
AT&H1&R2
```

Please refer to the list of supported modems (at the end of this paper) for details, or look for further details in the modem's user guide.

On the other end of the serial cable, the COM port of the PC must be set up as well to detect the state of signal CTS and to drive signal RTS. Both actions are achieved through the attribute "Flow control (FlowControl)" of the FAX Receiver / Sender:

```
FlowControl = "HW Handshake"
```

Note: Hardware flow control signals are exchanged in a transparent way between the modem and the UART driver, so that no application-level software is involved in such signal exchange. However, it is possible to track changes in the state of signal CTS during data transmission, for debug purposes (see attributes "Trace leve (TraceLevel)" and "Trace file (TraceFile)" inherited by class "FAX RECEIVER" from class "SERIAL RECEIVER").

Usually it is a good idea to keep hardware flow control enabled, however sometimes the CTS and RTS signals may not be available in the modem hardware. In such conditions, switch to software flow control alone (see section below).

10.1.5.2 Software flow control

Software flow control uses data in the communication stream to control the transmission and reception of data. Two special characters are used, XOFF (ASCII d'19') and XON (ASCII d'17'). When data flows form one device to the other, the receiver side issues an XOFF character to tell the other end to suspend transmission. XON will resume transmission afterwards.

The main advantage with software flow control, is that it does not need dedicated hardware in the serial cable.

The main disadvantages are first of all, a small loss of performance as additional control character are added to the original data stream. Moreover, it could happen that a binary data stream carrying such special characters as part of the information causes communication failure if the receiver end cannot take them as control characters.

As well as for hardware flow control, software flow control must be enabled on the modem side as well as on the PC's COM port.

For instance, the US Robotics 56k enables software flow control as follows:

```
AT&H2&I1
```

To initialize the COM port, the option "SW Handshake" or "Enabled" has to be selected via the attribute "Flow control (FlowControl)" (defined in the class "FAX RECEIVER").

Basically, for Output control (FAX send) the difference between these two settings is:

- With Flow control = "SW handshake" the COM port driver of the PC handles XON/XOFF characters transparently, i.e. the application-level software (Fax Receiver / Sender) does not even receive such control characters because they are filtered before reaching this point.
- With Flow control = "Enabled" the COM port driver of the PC does not handle XON/XOFF characters, so the application software (Fax Receiver / Sender) must do the job of suspending and resuming data transmission.

With input control (FAX Receive) the PC is normally so much faster than the modem in receiving data, so that flow control is normally useless - any settings should work.

10.1.6 Fax hang-up error list

The following list defines all possible error codes a fax modem can return to the host PC. The code and literal description are reported on the message window for logging.

Error Code	Description
0	Normal end of connection
1	Ring detected without connection
2	Call aborted
3	No loop current
10	Unknown phase A error
11	No answer within T1 timeout
20	Unknown phase B error
21	Remote cannot receive
22	COMREC error in transmit phase B
23	COMREC error invalid command received
24	RSPREC error in phase B
25	DCS sent three times without response
26	DIS received three times; DCS is not recognized
27	Failure to train at +FMS value
28	RSPREC invalid response received
40	Unknown phase C error
41	Unknown image format error
42	Image conversion error
43	DTE to DCE data underflow
44	setup transparent data
45	Incorrect line length in image
46	Incorrect page length in image
47	Incorrect compression code
50	Unknown phase D error
51	RSPREC error in phase D
52	No response to MPS after three attempts
53	Invalid response to MPS
54	No response to EOP after three attempts
55	Invalid response to EOP
56	No response to EOM after three attempts
57	Invalid response to EOM
58	Unable to continue after PIP or PIN
70	Unknown receive phase B error
71	RSPREC error in phase B transmit
72	COMREC error in phase B receive
73	T.30 T2 timeout expected; page not received
74	T.30 T1 timeout after EOM received
90	Unknown receive phase C error
91	Missing EOL after five seconds
92	Invalid CRC or FCS
93	DCE to DTE buffer overflow
A0	Unknown receive phase D error
A1	RSPREC invalid response received
A2	COMREC invalid response received
A3	Unable to continue after PIN or PIP

10.1.7 Fax Standards and Protocols

This section gives a short overview to the various standards defined for fax modems.

10.1.7.1 Group 3

Group 3 is the international standard for communication between two fax devices (fax machines or fax boards). Fax machines have evolved over the past 20 years. Groups 1 and 2 fax machines transmit a single page at six and three minutes respectively and were used throughout the 1970s. Group 3 transmits one page in as little as 20 seconds (at 9600 bps). Group 3 resolution is 203x98 dpi in standard mode and 203x196 dpi in fine mode. Virtually all fax machines sold in the market today are Group 3 units.

10.1.7.2 V.27ter

V.27ter is the modulation scheme used in Group 3 Facsimile for image transfer at 2400 and 4800 bps.

10.1.7.3 V.29

V.29 is the modulation scheme used in Group 3 Facsimile for image transfer over dial-up lines at 9600 and 7200 bps.

10.1.7.4 V.17

V.17 is a new CCITT standard. It is the modulation technique for use in extended Group 3 Facsimile that allows a fax transmission at a speed of 12000 and 14400 bps.

10.1.7.5 Class 1 & class 2

In the past no standard existed for a microcomputer to deal with a fax board. As a result, the software for a particular fax board won't work with another fax board from a different manufacturer. The Electronic Industries Association/Telecommunications Industry Association (EIA/TIA) developed new standards (class 1 & class 2) for microcomputers to communicate with fax modems. As a result new fax software will work with class 1 or class 2 fax modems of any manufacturer.

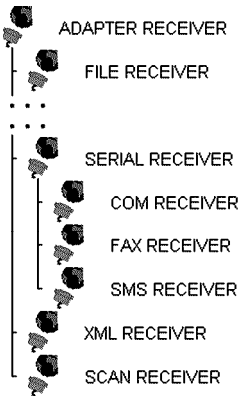
The class 1 standard provides minimal hardware support for sending a fax from a microcomputer, while class 2 adds over 40 AT-command set instructions and places more functionality into the modem.

class 2 standard was finalized after 1992.

Some modems support both the industry-standard FAX class 2 protocol and the EIA/TIA-592-A FAX class 2.0 standard. Both standards provide a similar level of functionality, although the commands and implementations are slightly different.

10.2 Components of the Fax Receiver / Sender

Instances of the class "FAX RECEIVER" can be used to send and / or receive fax messages via a fax modem connected to one of the serial COM ports. As depicted below, the class "FAX RECEIVER" is derived form the class "SERIAL RECEIVER" which has general attributes defined for all types of Receivers used as interface for peripherals connected to one of the serial COM ports.



10.2.1 Attributes of the base class "SERIAL RECEIVER"

COM port (ComPort)	This attribute is used to specify the serial port the fax modem is attached to (COM1-5).
Bit rate (BitRate)	This attribute is used to specify the speed at which the computer sends/receives data via the serial cable connected to/from the modem (1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200).
Parity (Parity)	This attribute is used to specify the type of parity checking for the transmitted data (No, Odd, Even, Mark, Space).
Data bits (DataBits)	This attribute is used to specify the number of data bits used (4-8).
Stop bits (StopBits)	This attribute is used to specify the type of stop bits (One, One half, Two).
Flow control (FlowControl)	The attribute is used to specify the type of flow control between modem and computer (Disabled, Enabled, HW Handshake, SW Handshake). It is recommended to select option "Enabled" as hardware signals are kept active, thus no process will stop other processes. If "HW Handshake" is selected, the general signals of the modem hardware are used to stop the computer from transmitting data if the modem is too slow. If "SW Handshake" is selected, the firmware of the modem sends byte 17 (Xoff) to the computer, if it is to slow to receive the data and byte 19 (Xon) when ready again.
Timeout on response [s] (Timeout)	This attribute is used to specify the timeout interval in seconds the Receiver waits for a response by the modem, a value between 12 and 20 seconds should be defined (0 = infinite, no timeout: to be used only for debugging).
Trace level (TraceLevel)	This attribute is used to specify the type of trace via a bit field, thus any combination can be specified (1=INFO, 2=ACTIONS, 4=SERIAL, 8=DUMP, 80=SRCINFO). In the trace line number and the source filename are output to be able to track processes at run time. By specifying 255, a trace with all available information is generated, by default specify 0 to suppress the generation of a trace.
Trace file (TraceFile)	This attribute is used to specify the path and filename of the trace file.

10.2.2 Attributes of the class "FAX RECEIVER"

Program (Program)

This attribute is overloaded from the base class "ADAPTER RECEIVER" and specifies the Papyrus Server Module used for the Fax Receiver (the string defined is linked to the respective Program object of the currently used operating system and contains path and filename of the executable).

BusyFlag (BusyFlag)

This attribute is overloaded from the base class "ADAPTER RECEIVER" with the value 1. When the Fax Receiver / Sender is started, it initializes the communication with the modem and no fax messages can be sent. Therefore this attribute should be set to 1 to avoid that an Agent for instance starts sending fax messages before the initializing process has finished.

Flow control (FlowControl)

This attribute is overloaded from the base class "SERIAL RECEIVER" and the option "HW Handshake" is defined as default.

Temporary folder (TmpDir)

This attribute specifies the directory used to store the temporary G3 files generated during the fax pages are received. If no directory is defined via this attribute, the temporary directory defined via the environment variable TEMP is used.

Local ID (LocalID)

Enter the own phone number starting with the country code (may only contain numbers). This number is used as an identification between fax machines and output in the fax header.

Display Msg (DisplayMsg)

Via this attribute the current Status of the Fax Receiver / Sender is output to give a visual feedback

Dial Prefix (DialPrefix)

This attribute specifies the number required to exit from the local switchbox (e.g. "0,", where the comma is interpreted as delay by the modem) to get an external line. By this it is not required to modify indices in the AFP containing phone numbers without this leading number. The contents of this attribute is automatically concatenated as leading part with the contents of the attribute "Dialing number".

Max Bit Rate (MaxBitRate)

This attribute is used to define the maximum bit rate the modem uses to transmit data to the modem on the other side (by default 14400 should be specified). By default the modems determine the maximum speed at handshake by starting with the highest speed and slowing down until a communication is possible. To avoid that the bit rate is set too high during handshake (e.g. the phone line gets noisy right afterwards), it is possible to define a lower bit rate for the modem to start at.

Rings (Rings)

This attribute specifies how many rings the Fax Receiver / Sender has to wait until it recognizes the call as FAX (0-6) and goes off-hook. If the number of rings is set to "0", the Fax Receiver / Sender will not answer incoming calls. By this it is possible to define that the phone line is dedicated for outgoing faxes.

Fax Header (FaxHeader)	If this checkbox is selected, a fax header is output containing fax number of sender, date, time, current page, and total number of pages.
Fax Logo (FaxLogo)	This attribute specifies a string that should be output in the header of the fax (e.g. "ISIS Software AG").

10.2.3 Attributes of the class "FAX MODEM"

An instance of the class "FAX MODEM" has to be added to the Fax Receiver / Sender containing general settings for the respective modem. Predefined templates for the supported modems can be found in the Library folder "Library / Fax Receiver/Sender / Fax modem templates".

Init string (InitString)	Enter the initialization string provided by the user's manual of the modem to define the command set used to initialize the modem.
Fax class (Faxclass)	Enter the fax class provided by the user's manual of the modem to define the type of modem (e.g. Elsa Microlink 56K is a fax class 2.0 modem).
Xon expected (XonExpected)	If this checkbox is selected, the Xon / Xoff flow control is used (by default deselected).
Rx bit order reversed (RxBitOrderReversed)	If this checkbox is selected, the bit order of the received bytes is reversed during Fax Receive Phase C.

10.2.4 Attributes of the class "FAX"

A template of the class "FAX" has to be added to the Task template used for inbound fax messages, thus the "Fax Adapter" is able to fill the attributes with values of the received fax message. The required match criteria in the field "Type property" of each of the following attributes are listed in section *"Mapping fax data into attributes"* of the following chapter *"Usage of the Fax receiver sample"*.

Fax number (FaxNumber)	The fax number of the sender of the fax message.
Fax pages (FaxPages)	Number of pages of the received fax message.
Start date (StartDate)	Start date of the fax message.
Start time (StartTime)	Start time of the fax message.
End date (EndDate)	End date of the fax message.
End time (EndTime)	End time of the fax message.
Transmission complete (TransmissionComplete)	Flag which is set if the complete fax message has been received successfully.
Date (Date)	Additional information, currently not in use.

10.2.5 Attributes of the Tool class "FAX TIFFG3"

Index name (FAX3FaxIndex) Enter the name of the Group Index defined in the AFP file containing the fax number of the recipient. The "Adapter (TIFFG3)" maps the fax number into the attribute "FaxNumber (FaxNumber)" of the Material "Send fax" when instantiating the Task template "Send fax to" into the "Send fax queue".

10.2.6 tool/material method of Tool class "FAX TIFFG3" and Material class "FAX AFP"

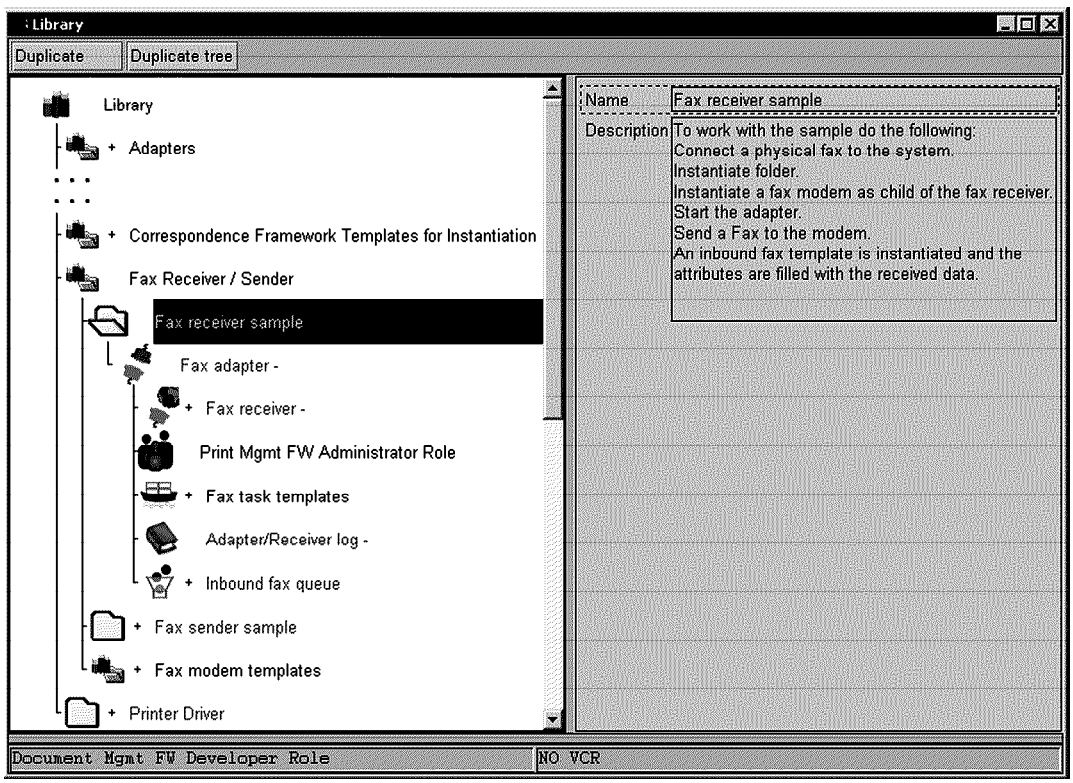
Fax AFP (StartFaxAFP) Via the expression FAX3FaxIndex==%FAX3FaxIndex% the fax number defined in the attribute "Index name (FAX3FaxIndex)" is mapped into the method parameter FAX3FaxIndex.

10.2.7 tool/material method of class "FAX RECEIVER" and Material class "FAX AFP"

Send Fax (StartSendFax) Via the Tool method parameter "FaxNumber=FaxNumber==" the fax number, which has been copied from attribute "Index name (Fax3FaxIndex)" of the Tool "Fax TIFFG3" into the attribute "FaxNumber (FaxNumber)" of the Material "Send fax", this tool/material method is able to send the fax to the appropriate addressee.

10.3 Usage of the Fax receiver sample

After connecting a fax modem to one of the serial COM ports, instantiate the folder "Fax receiver sample" and add an instance of the appropriate Fax Modem template as child object of the Fax Receiver / Sender. Select the "Fax adapter" and click the button "Start adapter" in the toolbar. By this the Adapter starts its Fax Receiver / Sender automatically. Release each Queue by selecting it and clicking the button "Release Queue", which automatically starts the Agent in each of these Queues.


Library
Duplicate
Duplicate tree
Library
+ Adapters
+ Correspondence Framework Templates for Instantiation
Fax Receiver / Sender
Fax receiver sample
Fax adapter -
+ Fax receiver -
Print Mgmt FW Administrator Role
+ Fax task templates
Adapter/Receiver log -
+ Inbound fax queue
+ Fax sender sample
+ Fax modem templates
+ Printer Driver
Name
Fax receiver sample
Description
To work with the sample do the following:
Connect a physical fax to the system.
Instantiate folder.
Instantiate a fax modem as child of the fax receiver.
Start the adapter.
Send a Fax to the modem.
An inbound fax template is instantiated and the attributes are filled with the received data.
Document Mgmt FW Developer Role
NO VCR

10.3.1 Modem initialization

After starting the Adapter, the Fax Receiver / Sender should be able to initialize the modem. Open the message panel to observe if anything goes wrong in this phase. If the modem responds correctly to Hyper Terminal, and if the same communication settings are used for the Fax Receiver / Sender, no issues are normally expected during modem initialization. However, it is possible that the connected modem only supports fax class 1 (instead of 2 or 2.0), thus the modem initialization fails. If the initialization was successful, the state of the Fax Receiver / Sender changes from "Busy" to "Ready".

10.3.2 Receiving of fax messages

As soon as the fax modem is initialized the Adapter changes its state from "Busy" to "Ready" and the Fax Receiver starts listening on the serial line for incoming fax messages. The Fax Receiver / Sender transfers the data of incoming fax messages to the Adapter which instantiates the appropriate Inbound Fax template in its Queue "Inbound fax queue" and stores the fax as multi-page TIFF G3 in the Data object "Fax image" which can be viewed by double clicking it.

After the fax modem has been initialized, the Fax Receiver / Sender remains listening on the serial line for incoming calls. When an incoming call is detected, the Fax Receiver / Sender changes its state from "Ready" to "Busy" to prohibit potential conflicting requests by the operator (e.g. attempt to send a fax while another one is being received).

The Fax Receiver / Sender reads incoming fax data from the line, and stores each received page as a G3 file (*.g3) in the temporary directory specified via the attribute "Temporary folder (TmpDir)" of the Fax Receiver / Sender. Once a fax document is received and the session is closed (successfully or not), the temporary G3 files (one or more) are concatenated converted into a TIFF binary object.

When a fax call arrives, the Fax Receiver / Sender performs the following actions in the given sequence:

1. **Phase A** - ***Answer the call***
   An incoming call is announced by a "RING" command string issued by the modem receiving the call and send to the host PC. The attribute "Rings" of the Fax Receiver / Sender determines how many "RING" signals the Fax Receiver / Sender has to wait (default = "1") before going off-hook.

2. **Phase B** - ***Negotiate session parameters***
   At this stage the modem communicates with the caller station and negotiates parameters for sending and receiving the fax message (e.g. polling, type of data, transmission speed, resolution, width, etc.).

   When the receiving modem goes off-hook, the first action is to recognize the fax call (in contrast to a possible voice / data calls). The caller station typically sends an 1,100 Hz tone and the called station responds with a 2,100 Hz tone. If one station does not receive the tone from the other one, or in case of any Phase B failure, the session is aborted.

   To debug Phase A and B it is possible to turn on the modem speaker via the parameters "L*n*" and "M*n*" defined in the initialization string sent to the modem. The meaning can vary slightly depending on the model, but in general the settings are as follows:

Parameter	Description
L0	speaker volume low
L1 or L2	speaker volume medium
L3 or L4	speaker volume high
M0	speaker always OFF
M1	speaker ON until end of Phase B
M2	speaker always ON

   During Phase B, the receiving modem identifies its capabilities to the transmitter, and the parameter values used depend on the capabilities of the receiving modem. Older models for instance can achieve 9600 bps, while recent ones can run at 14400 bps.

The amount of noise on the phone line during handshake can also impact the transmission speed. Peers will try to communicate a known data pattern starting with the highest bit rate possible, and based on the error rate reported from the receiver side, they retry the communication at lower rates until the signal is received free of errors. However, if the line is still too noisy at 2400bps (the lowest bit-rate possible), fax transmission ends and the session fails.

The amount of noise on the phone line can depend on the specific connection. Being aware of this, fax callers are usually programmed to re-dial the same number a few seconds or minutes later. In most cases this approach allows to successfully transmit the whole fax document.

3. **Phase C** - ***Receive one page of fax document data***
The fax document page is transmitted based on the parameters negotiated between the caller station and called applications in Phase B.

This phase is critical from a timing point of view. The PC must be fast enough to receive (read) data flowing from the local modem, otherwise a buffer overrun might occur on the PC's UART and the resulting fax data will be corrupted or Phase C fails at all. For this reason, it is important that the CPU is not overloaded by too many applications, while the Fax Receiver / Sender is running. The local line speed must be greater than the line speed of the incoming fax call (14400 bps). A bit rate of 19200 bps for instance could be enough, but to prevent random fax transmission errors the margin should be bigger, therefore 57600 bps is the recommended value (some modems also suggest 115200 bps).

According to the fax protocol implementation, the actual G3 data is preceded by a null byte (x'00'). This is why the Fax Receiver / Sender discards any data before the null byte is received and generates a new temporary G3 file to store the received data.

G3 files are stored in the directory specified via the attribute "Temporary folder (TmpDir)" of the Fax Receiver / Sender. If the folder is not defined or not existing, the temporary folder definded via the environment variable TEMP is used instead.

The name of the temporary G3 files consists of the prefix "fax", a 4 digit PID (hex value) identifying the fax receiver process, a 4-digit progressive number (from 0000 to FFFF, if no more numbers are available the error message "Unable to create file ..." is issued), and a decimal number ranging from 1 to 99 which indicates the page number.

G3 files are normally deleted from disk when fax receive session ends. However, it is possible to leave the files on disk by setting the parameter "KeepTemporaryFiles". In this case, the size of the temporary folder shall be kept under control by an external script (if not manually).

During Phase C, the bit order of the data bytes flowing from the modem to the PC depends on the modem type. Some modems automatically reverse the PSTN facsimile data arriving from the data carrier, so that the first bit received from the carrier is the last bit sent to to the PC. The Fax Receiver / Sender corrects this difference by means of an initialization command that reverts bit order when needed.

The following two codes are supported:

+(F)BOR=0	Selects direct bit order for Phase C data.
+(F)BOR=1	Selects reversed bit order for Phase C data.

The attribute "Rx bit order reversed (RxBitOrderReversed)", defined in the class "FAX MODEM", determines the setting for the connected model.

4. **Phase D** - ***Acknowledge received page***
   After a fax page has been transmitted a continuation value is used to indicate to the receiving modem what to do next. The following events may take place:
   The caller modem has no more pages to send, in this case Phase E starts.
   The caller modem has at least one more page to send, in this case Phase B starts over.
   The called modem failed to receive the last page. In this case the session is aborted, a fax binary object is instantiated but the job goes to Error and the flag "Transmission Completed" is set to false.

5. **Phase E** - ***Release the fax call***
   As soon as the complete fax message has been transmitted, the caller station sends a disconnect signal and both fax stations disconnect from the phone line.
   On the receiver side, the G3 files collected during Phase C are concatenated and a TIFF header is added to each page in order to generate a TIFF file (multi-page, if required). The TIFF file is then passed to Papyrus Objects as a binary object. The attribute "Transmission Completed" is set to true in the Fax Material.

10.3.3 Troubleshooting

During each phase, some errors can occur that may cause the session to fail (i.e. terminate prematurely). This section describes the most frequent errors and malfunctions that have been reported.

10.3.3.1 Phase A

To verify that an incoming call is a fax transmission, the modem waits for the 1,100 Hz tone sent by the caller station. As it may happen that the incoming call is not a fax transmission but a voice call, the 1,100 Hz tone would never be sent by the caller station. Therefore a timeout has to be defined via the Register S7, which should be ranging from 60 to 90 seconds.

The predefined templates of the class "FAX MODEM" already have this timeout defined in the initialization string (attribute "Init string (InitString)" of the respective Modem object derived from the class "FAX MODEM"), to define the appropriate timeout interval for a modem of a different manufacturer, customize the initialization string according the specifications given in the user's manual of the modem.

10.3.3.2 Phase B

Any error occurring during Phase B will cause the complete loss of the current fax document page, nevertheless preceding pages will be instantiated as a partially received fax binary object in the Data object "Fax image" of the Fax Task, whereas the flag "Transmission Completed" is set to "False".

If the phone line is very noisy, Phase B may fail more often than usual. In such a case a quick way to determine the line quality is to plug a standard telephone in place of the fax modem and to listen how the signal quality varies within a phone call and in subsequent calls. If a lot of static noise is detected, exchange the cable and check if the sound quality improves.

10.3.3.3 Phase C

During Phase C, if an error occurs some data has probably been received, so a partially received page will most of the times be instantiated. The most frequent reasons for Phase C errors can range from randomised noise on the line to an overloaded host CPU (which may cause a buffer overrun).

Check the bit-rate on the local line (must not be below 19200 bps; 57600bps is recommended). If the data is corrupted, it is possible that the attribute "Rx bit order reversed (RxBitOrderReversed)" is not set correctly in the instance of the class "FAX MODEM" added to the Fax Receiver / Sender.

10.3.3.4 Phase D

Phase D errors are not very frequent and in most cases they are just Phase C errors which are detected by the modem too late (when the page ends). The resulting page could be incomplete or corrupted in such cases.

10.3.3.5 Phase E

A possible Phase E error is that the hard disk is full, thus there is not enough space left for the TIFF file creation. This is a rare but fatal event that causes loss of data (see also parameter "KeepTemporaryFiles").

Free space in the temporary directory (specified via the attribute "Temporary folder (TmpDir)" or by default the path specified via the environment variable TEMP) if needed.

Phase E errors related to phone line or modem issues normally do not inhibit the full completion of the received fax, despite the attribute "TransmissionCompleted" flag remains unchecked and the Tool changes its state to "Error" instead of "Completed".

10.3.4 Instantiation of the appropriate Task template

To enable the "Fax Adapter" to instantiate the Task template, a match criterion with one of the keywords ($MATCH_RECURSIVE$, $MATCH_LOCAL$ or $MATCH_ITERATION$) is required in the field "Type Property" of one of the Task attributes. Due to the fact that the modem already verifies that an incoming call is a fax message, the Receiver automatically generates the name / value pair "Match=FAX" each time a fax message is received. As the attribute "Match" of the Task template "Inbound fax from" has the match criterion **"FAX($MATCH_RECURSIVE$,Match)"** defined in its field "Type property", the Adapter will compare the content of this attribute with the value of the name / value pair "Match=FAX" transmitted by the Receiver. Since both are equal the Adapter instantiates the Task template into the "Inbound fax queue".

10.3.5 Mapping Fax data into attributes

The following match criteria are defined in attributes of the child objects of the Task template "Inbound fax from", thus the "Fax Adapter" will fill them with the respective data:

Fax object "FAX"

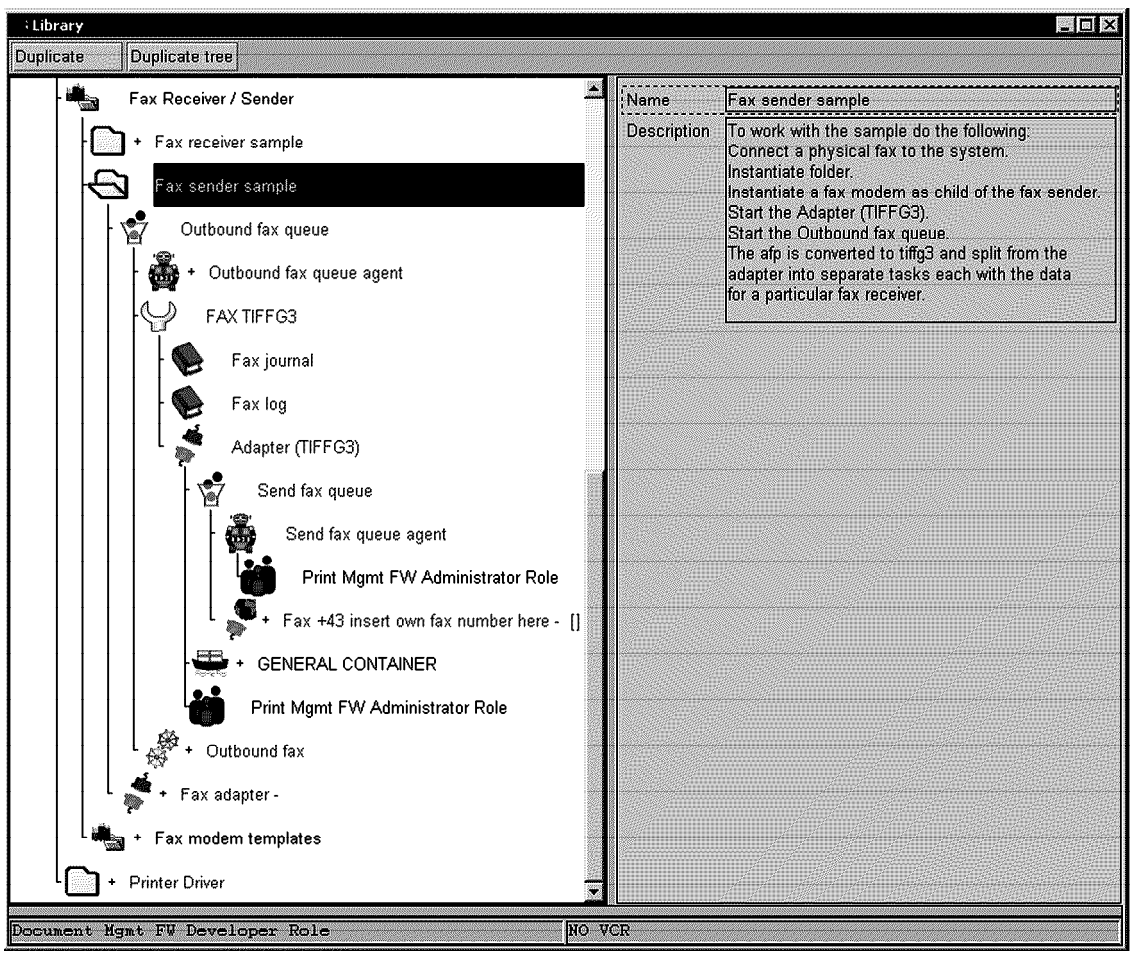
attribute	Type property
Fax number (FaxNumber)	FAX(RemoteId)
Fax pages (FaxPages)	FAX(FaxPages)
Start date (StartDate)	FAX(StartDate)
Start time (Start time)	FAX(StartTime)
End date (EndDate)	FAX(EndDate)
End time (EndTime)	FAX(EndTime)
Transmission complete (TransmissionComplete)	FAX(TransmissionCompleted)

Data object "Fax image"

attribute	Type property
Fax name (TargetName)	MIME(image/tiff);FAX(TargetName)

10.4 Usage of the Fax sender sample

After connecting a fax modem to one of the serial COM ports, instantiate the folder "Fax receiver sample" and add an instance of the appropriate Fax Modem template as child object of the Fax Receiver / Sender (regardless to which reference of the Fax Receiver / Sender). Start the "Adapter (TIFFG3)" (used to instantiate the appropriate Fax Task template) and the "Fax adapter" (used to start the Fax Receiver / Sender which sends the fax message) by selecting each of them and clicking the button "Start adapter" in the toolbar. By this the Fax Receiver / Sender is started automatically. Release each Queue by selecting it and clicking the button "Release Queue", which automatically starts the Agent in each of these Queues.


Library
Duplicate
Duplicate tree
Fax Receiver / Sender
Fax receiver sample
Fax sender sample
Outbound fax queue
Outbound fax queue agent
FAX TIFFG3
Fax journal
Fax log
Adapter (TIFFG3)
Send fax queue
Send fax queue agent
Print Mgmt FW Administrator Role
Fax +43 insert own fax number here - []
GENERAL CONTAINER
Print Mgmt FW Administrator Role
Outbound fax
Fax adapter -
Fax modem templates
Printer Driver
Name
Fax sender sample
Description
To work with the sample do the following:
Connect a physical fax to the system.
Instantiate folder.
Instantiate a fax modem as child of the fax sender.
Start the Adapter (TIFFG3).
Start the Outbound fax queue.
The afp is converted to tiffg3 and split from the adapter into separate tasks each with the data for a particular fax receiver.
Document Mgmt FW Developer Role
NO VCR

10.4.1 Modem initialization

After starting the "Fax adapter", the Fax Receiver / Sender should be able to initialize the modem. Open the message panel to observe if anything goes wrong in this phase. If the modem responds correctly to Hyper Terminal, and if the same communication settings are used for the Fax Receiver / Sender, no issues are normally expected during modem initialization. However, it is possible that the connected modem only supports fax class 1 (instead of 2 or 2.0), thus the modem initialization fails. If the initialization was successful, the state of the Fax Receiver / Sender changes from "Busy" to "Ready".

10.4.2 Sending of fax messages

The sender part of the Fax Receiver / Sender is implemented as a Tool method in the class "FAX RECEIVER" with the corresponding Material method defined in the class "CONVERT FAX".

The Task "Outbound fax" in the example above is used as a carrier for an AFPDS file, which has to be converted into a multi-page TIFF G3 required by the Fax Receiver / Sender to send it as a fax message. Media types other than AFP files can also be sent, the only requirement is a converison into the required TIFF G3 format.

The Tool method "Fax AFP (StartFaxAFP)" (inherited by the Tool "Fax TIFFG3"), used in combination with the Material method "Fax AFP (StartFaxAFP)" (inherited by the Material "FAX AFP" within the Task "Outbound Fax"), generates the required multi-page TIFF G3 and passes it to the "Adapter (TIFFG3)" as input.

The "Adapter (TIFFG3)" instantiates the Task template "Send fax to" from its General Container into the Queue "Send fax queue" and the Fax Receiver therein is used as a Tool to send the fax message. The Tool method "SendFax (StartSendFax)" (inherited by the Fax Receiver / Sender "Fax +43..."), used in combination with the Material method "SendFax (StartSendFax)" (inherited by the Material "Send fax" of the Task "Send fax to"), sends the fax message using the modem settings defined in its child object derived from the class "FAX MODEM".

Note: The fax number of the Fax Receiver / Sender can be changed manually in the Material "Send fax" of the Task template "Send fax to".

After initialization of the modem, the Fax Receiver / Sender waits for a match to occur. Incoming fax calls can be disabled by setting the attribute "Rings" of the Fax Receiver / Sender to "0". In this case, incoming calls will not be answered, but still the Fax Receiver / Sender will stay in state "Busy" as long as the line rings, thus fax transmission is not possible during this time. During fax transmission the state of the Fax Receiver / Sender changes its state from "Ready" to "Busy" to prohibit sending attempts while the current fax message is not yet sent.

The Fax Receiver / Sender performs the following actions in the given sequence:

1. ***Check the TIFF G3 and attributes***
   As soon as the tool/material method "SendFax (StartSendFAX)" is invoked, the following actions take place:
   - The fax number to dial is the result of the concatenated strings found in the attribute "Dial Prefix (DialPrefix)" (defined in the class "FAX RECEIVER") and the attribute "FaxNumber (FaxNumber)" (defined in the Material class "CONVERT FAX"). Leading spaces in the attribute "FaxNumber (FaxNumber)" are automatically removed. If for instance DialPrefix = "0" and FaxNumber = " 001 123 456" (three leading spaces) the number to dial woud be "0001 123 456".
   - The TIFF G3 file is scanned for each page start/end points. Most of the TIFF file format errors are detected in this phase, i.e. before the modem goes off-hook. This approach helps to speed-up the recovery phase.
   - The contents of the attributes "Resolution (Resolution)" (defined in the template "FAX (BINARY)", can be set to fine or normal), "From page (PageFrom)", and "To page (PageTo)" (both defined in the class "MATERIAL"), used during subsequent phases, are parsed.
   - The attribute "Retry counter (RetryCounter)" (defined in the class "CONVERT FAX" inherited by the template "Send fax") is incremented to keep track of the number of transmission attempts made by the Fax Receiver / Sender. This value can be used for a retry mechanism implemented in Papyrus objects.

2. **Phase A** - ***Dial the fax number***
   The number to dial must contain valid ASCII characters. The set of valid characters can vary slightly depending on the modem model. For instance a **"+"** sign to mark international calls (resolving to double zero "00") may not be valid on certain modem models. The same applies for a comma "," used to add a pause within digits (Register S8 defines the duration of the pause in seconds) and a hyphen "-" to separate digits. However, digits "0" - "9" are obviously valid on all models and a white space is generally accepted to separate groups of digits.

3. **Phase B** - ***Negotiate session parameters***
   At this stage the Fax Receiver / Sender negotiates the values of the parameters for the data transmission in Phase C (for details see description of Phase B in section "Receiving of fax messages" above).

4. **Phase C** - ***Send one page of fax document data***
   The fax document page is transmitted based on the parameters negotiated between the caller and called applications in Phase B (see also description of Phase C in section "Receiving of fax messages" above).

   When Phase C starts, the following operations are performed:
   - The file read pointer is positioned to the beginning of the next G3 data block (start of page).
   - G3 data is read on a byte by byte basis. Some bytes have special meaning to fax protocol so they are filtered before the transmission, e.g. DLE (octal 20) is transmitted as DLE DLE, DLE DLE is transmitted as DLE SUB (octal 32).
   - When an XOFF character (octal 23) is received, the transmission must be temporarily interrupted.
   - When an XON character (octal 21) is received afterwards, the transmission must be resumed.

     Note that despite the type of flow control selected for COM port settings at start-up, Phase C always implements software flow control to avoid a buffer overflow in the caller modem's buffer. This means, that during G3 data transmission, flow control bytes are sent by the modem to the host PC.

     For some modem models, an XON character marks the start of the transmission, thus the sending modem has to wait for such a character before starting to send the G3 data of each page. This setting is defined in the attribute "Xon expected (XonExpected)" in the Modem specification addes to the Fax Receiver / Sender (derived from the class "FAX MODEM").

     Some modem manufacturers recommend to use hardware flow control in addition to software flow control. In this case, the attribute "FlowControl" of the Fax Receiver / Sender must be set to "HW Handshake".

5. **Phase D** - ***Acknowledge page transmission***
   At the end of Phase C an end-of-page marker is transmitted by the caller modem to inform about page continuation. This procedure differs slightly depending on the fax class in use. In FAX class 2, a fixed page termination sequence is transmitted (DLE ETX) and a frame type description command is sent (AT+FET). In Fax class 2.0 the page termination DLE EOP defines the end of session and DLE MPS defines page continuation (i.e. another fax document page follows). The modem's response determines whether the page has been successfully received by the called modem or not.

6. **Phase E** - ***Release the fax call***
   In this phase, the transmitter end goes on hook and the session ends.

10.4.3 Troubleshooting

During each phase, some errors can occur that may cause the session to fail (i.e. terminate prematurely). This section describes the most frequent errors and malfunctions that have been reported.

10.4.3.1 Checking of TIFF G3 and attributes

If a format error is detected in the TIFF file the send session is aborted and the Material changes its state to "Error". A typical format error occurs if a multipage TIFF header is not byte-aligned with the G3 data section at the end of the previous page.

10.4.3.2 Phase A

The most frequent error occurring in this phase is the absence of a carrier signal on the line, which prevents the modem from dialling the number. Possible reasons for this error are:

Connection problems	Switch on the speaker of the modem to hear the line's tone. If there is no tone after dialling, maybe the wire is not properly connected or a prefix is required to access the external line (specified via the attribute "Dial Prefix (DialPrefix)" inherited from the class "FAX RECEIVER").
Number dialed to fast	If the number is dialed too fast to be recognized by the switch box, add a comma "," between digits or use the registers of the modem to increase the duration of each pulse / tone (refer to the manual of the modem for a list of registers to address). Differences may exist between nations and maybe within companies.
Dial tone or pulse	Check the phone company's requirements and set the initialization string accordingly (attribute "Init string (InitString)" of the respective Modem object derived from the class "FAX MODEM").
Carrier expected	The modem waits for a carrier but no carrier is expected. Disable "carrier detect" in the initialization string (attribute "Init string (InitString)" of the respective Modem object derived from the class "FAX MODEM").

10.4.3.3 Phase B

Several failure conditions can occur when the number has been dialed. The most frequent cases include:

- line is busy
- no response
- the number does not exist (no dial-tone)
- the number connects to voice or data communication

In all the cases above, the caller modem times-out after a given interval (typically 60 to 90 seconds) and another (one or more) attempts can be made thereafter according to Register S7 defined in the initialization (attribute "Init string (InitString)" of the respective Modem object derived from the class "FAX MODEM").

10.4.3.4 Phase C

The most frequent error in Phase C is a buffer underrun / overrun of the modem due to incorrect communication settings on the host side or due to an overloaded CPU during the transmission.

A possible test in such cases, is to reduce the maximum bit rate for transmission, i.e. enter a value lower than 14400 in the attribute "Max Bit Rate (MaxBitRate)" of the Fax Receiver / Sender. Another way to fix this problem is to change the attribute "Flow control (FlowControl)" of the Fax Receiver / Sender from "HW Handshake" to "Enabled" or vice-versa and to adjust the initialization string of the modem accordingly (for details refer to the manual of the modem).

10.4.3.5 Phase D

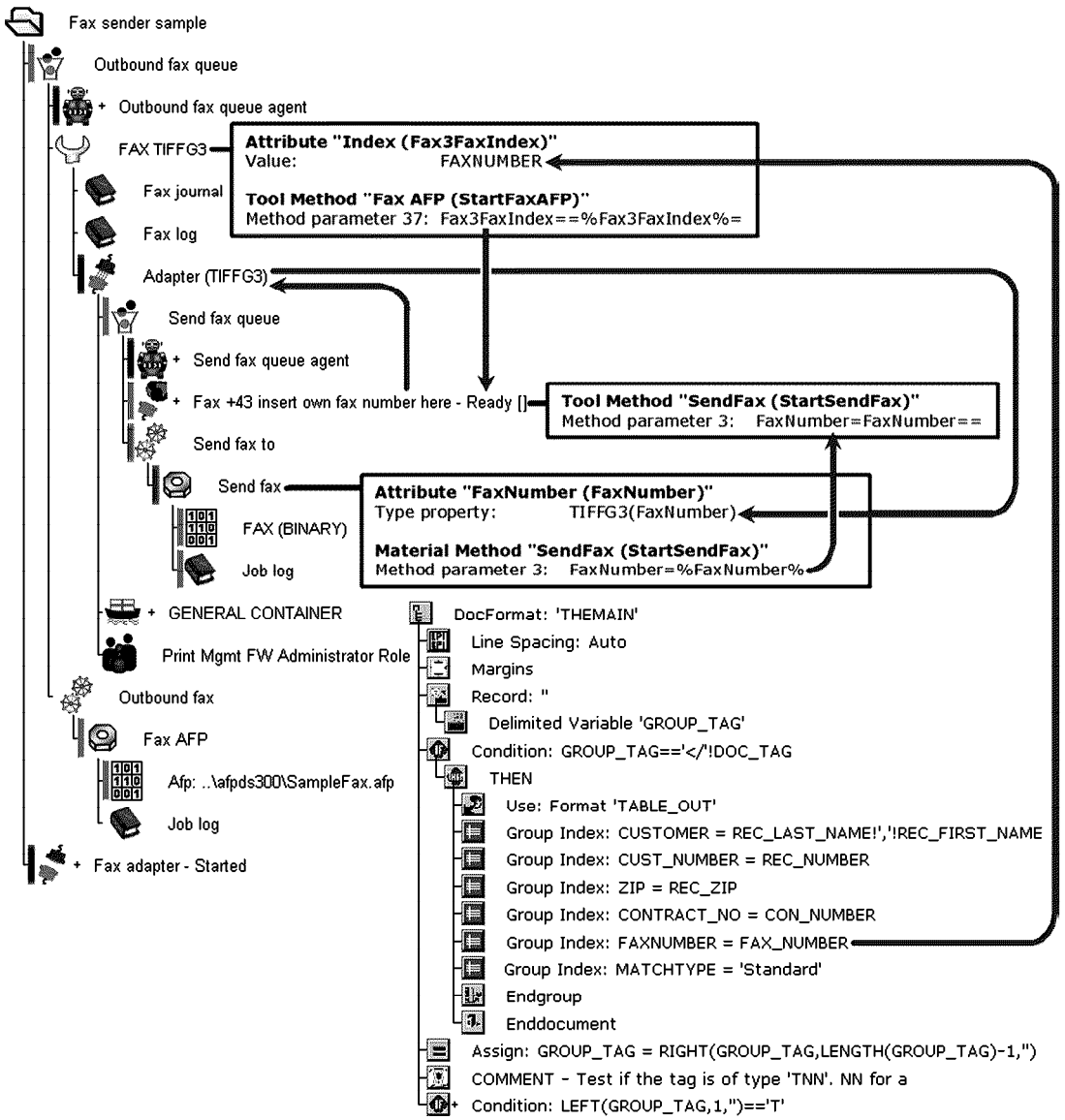
The modem's answer to an end-of-page sequence can be an error code if something went wrong during Phase C. Refer to Phase C troubleshooting above.

10.4.3.6 Phase E

Usually no errors are encountered during Phase E. However, it is possible that the fax has been successfully delivered despite the reported error.

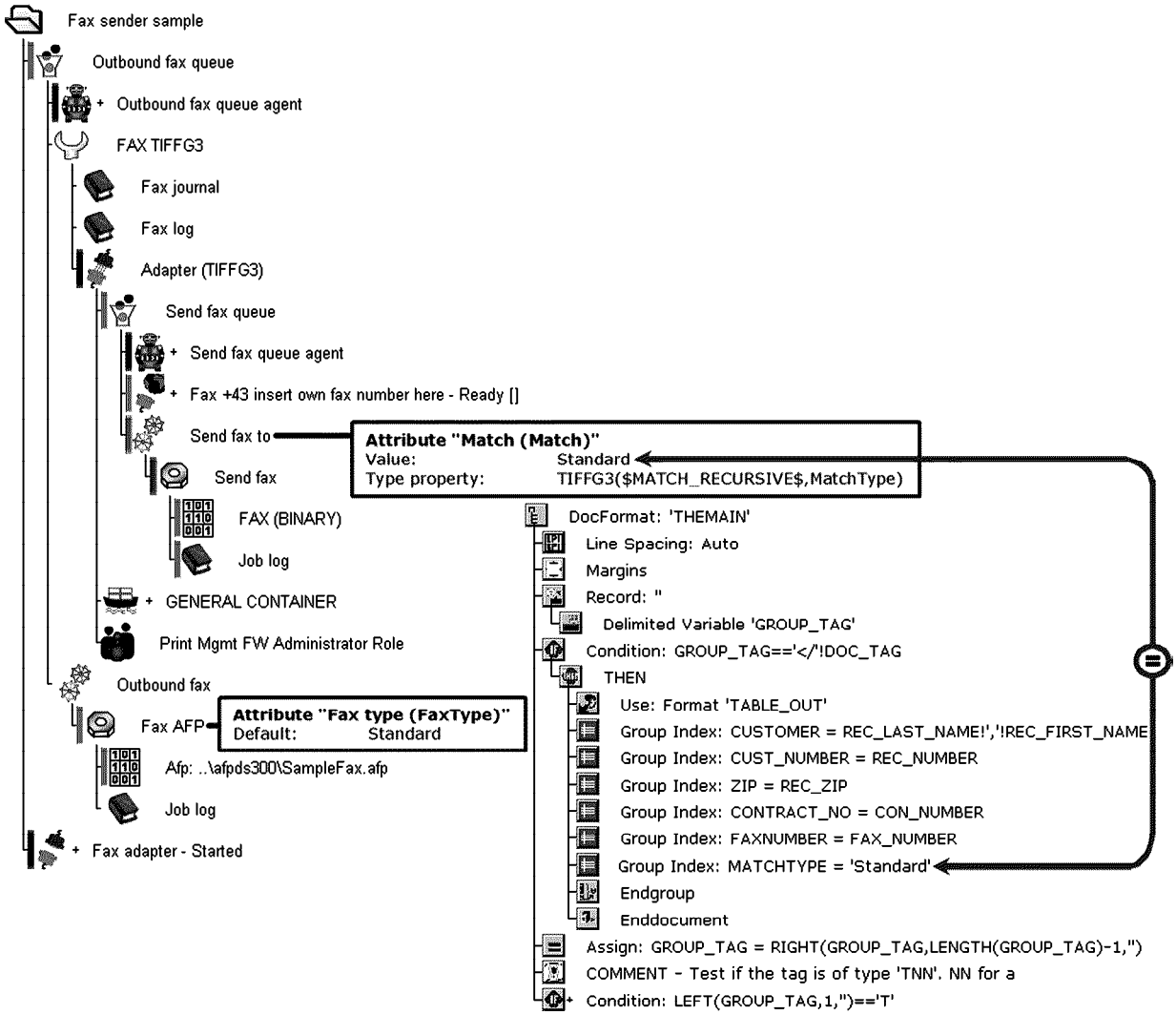
10.4.4 Using a fax number defined in the AFP

If the AFP has a Group Index defined containing the fax numbers for each document, the Tool "FAX TIFFG3" is able to retrieve it, by assigning its attribute "Index (Fax3FaxIndex)" with the name of the Group Index. Via the method parameter "Fax3FaxIndex==%Fax3FaxIndex=" in the Tool method "Fax AFP (StartFaxAFP)", the fax number is forwarded to the Fax Receiver / Sender which generates a name / value pair containing the fax number. The Adapter assigns the attribute "FaxNumber (FaxNumber)" with the fax number via the match criterion "TIFFG3(FaxNumber" in its field "Type property" and the content of this attribute is mapped into the method parameter "FaxNumber=FaxNumber==" of the Fax Receiver / Sender.



10.4.5 Instantiation of the appropriate Task template

To enable the "Adapter(TIFFG3)" to decide which Task template should be instantiated a match criterion with one of the keywords ($MATCH_RECURSIVE$, $MATCH_LOCAL$ or $MATCH_ITERATION$) is required in the field "Type Property" of one of the Task attributes. The Tool "FAX TIFFG3" writes the string found in the Group Index "MATCHTYPE" of the AFP into the attribute "Fax type (FaxType)" of the Material "Fax AFP", which is located in the Task "Outbound Fax". The default value "Standard" is defined in the field "Default" of the attribute "Fax type (FaxType)" of the Material template "Fax AFP" for the case that the Group Index "MATCHTYPE" cannot be found in the AFP. The Tool "Fax TIFFG3" passes the data of the Task "Outbound Fax" to the Fax Sender, which generates a name / value pair with the Group Index "MATCHTYPE" (e.g. MatchType=Standard or MatchType=Fine) and forwards it to the Adapter. As the attribute "Match" of the Task template "Send fax to" has the match criterion **"TIFFG3($MATCH_RECURSIVE$,MatchType)"** defined in its field "Type property", the Adapter will compare the contents of this attribute with the value of the respective name / value pair transmitted by the Receiver. If both are equal the Adapter instantiates the Task template into the "Send fax queue".



10.4.6 Mapping Fax data into attributes

The following match criteria are defined in attributes of the child objects of the Task template "Send fax to", thus the "Adapter (TIFFG3)" will fill them with the respective data:

Material "Send fax"

attribute	Type property
FaxNumber (FaxNumber)	TIFFG3(FaxNumber)

Data object "FAX (BINARY)"

attribute	Type property
Pages (CurrentPage)	TIFFG3(CurrentPage)
Resolution (Resolution)	TIFFG3(FaxResolution)
Fax name (FaxName)	MIME(image/tiff);TIFFG3(FaxContent)

11.0 HTTP Receiver / Sender

The HTTP Receiver has three distinct functions:

- It acts as a "Web Server" to deliver data over HTTP to applications such as web browsers. It currently supports HTML, XML, GIF, JPEG, TIFF, and PDF.
- It can receive data sent via the POST method (e.g. from HTML forms), and fill the data into object attributes.
- It can also send back a response, which makes it possible to create documents online from the transmitted data and send them back to the browser (e.g. as PDF).

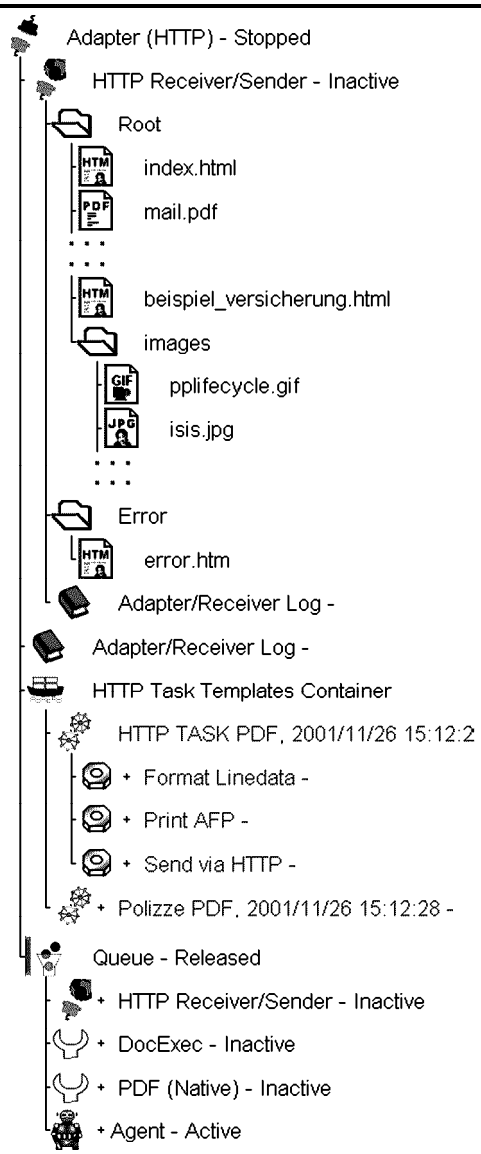


Figure 7: Typical setup for a HTTP Receiver / Sender

11.1 Object Structure

The object structure required for the HTTP Receiver is the same as for all other receivers. The Receiver must be the child of an Adapter and the siblings of the Receiver (i.e., the other children of the Adapter) must be a General Container holding the Task templates to be instantiated, and a Queue into which the Tasks are instantiated.

11.2 Basic handling of the HTTP Receiver / Sender

Before starting the HTTP Receiver / Sender, make sure it is configured correctly. In the Adapter, the *attribute "Type"* should contain the string *HTTP*. In the HTTP Receiver / Sender, the *attribute "Port"* must be set to the *TCP/IP port* on which the Receiver receives or sends HTTP requests. The default port is *80*. In order to start the HTTP Receiver, run the *method "Start Adapter"* on the Adapter. The Adapter will automatically start its associated Receiver / Sender. When the HTTP Receiver is started, it waits for incoming HTTP requests on the specified port. Although the functionalities of the HTTP Receiver are described separately here, the Receiver provides its functionalities at the same time, i.e. it must not be started in a certain mode. Instead, it uses the incoming request type (GET or POST) to decide what to do.

11.3 Web Server Functionality

When used as a "Web Server", a GET request is sent to the Receiver / Sender. Normally a web browser is used to send such a request. The Receiver / Sender then looks for the Request-URL of the GET request in its folder. This folder must be a child of the HTTP Receiver / Sender and it must have an internal name of "ROOT" (see the image above). The Receiver / Sender interprets the object structure below ROOT in the same way that normal web servers interpret a directory structure.

Example:

- Type in the URL http://hostname/index.html in the web browser (where hostname is the name of the computer where the HTTP Receiver / Sender runs).
- The browser generates a GET request looking like GET /index.html HTTP/1.1 and sends it to the port 80 of the computer called "hostname". "/index.html" is the Request-URL.
- The HTTP Receiver / Sender on this computer accepts the request and looks for a child object of the ROOT object which has a visible name (not internal name!) of "index.html".
- URLs are not case-sensitive, so the object could also be called "INDEX.HTML" or "Index.html".
- If such an object is found, the HTTP Receiver / Sender looks for an attribute that contains an appropriate MIME-Type via the function MIME in its "Type property" (e.g. MIME(text/html)).
  - If this attribute is a String attribute, its value is interpreted as a file name, and the contents of this file are sent back to the requester (the web browser in our example).
  - If the "TargetName" attribute is a binary attribute, the Receiver / Sender simply sends back its contents.
- If no matching object is found, the Receiver uses the object called "error.htm" in its ERROR child folder instead and sends it back in the same way as normal objects.
- This object could contain an HTML page telling the requester that the specified URL is not valid, or whatever is appropriate for the application.
- If one types in something like http://hostname/images/isis.jpg, the HTTP Receiver / Sender would look for an object called "isis.jpg" which is a child of an object called "images" (which in turn must be a child of "ROOT").

The MIME type of a returned object is currently derived from the "extension" of its visible name, i.e. from the part of the name which follows the last dot.

Important MIME Types:

Extension	MIME Type
.htm, .html	text/html
.xml	text/xml
.gif	image/gif
.jpg	image/jpeg
.tif	image/tiff
.txt	text/plain
.pdf	application/pdf

11.4 Receiving POST Data

The second functionality of the HTTP Receiver / Sender is to receive data sent via the HTTP POST method, and optionally returning an answer to the sender. A typical application would be to send HTML form data to the Receiver which is then used to construct an appropriate response. In the sample above, simple text is sent from a browser form. The HTTP Receiver / Sender receives it, DocEXEC creates an AFPDS from this text, the PDF converter converts it to PDF, and the Receiver / Sender sends back this PDF. Of course, the HTTP Receiver / Sender does not care who sends the POST data. It does not have to be a web browser; any application capable of producing an HTTP compliant data stream can communicate with the HTTP Receiver / Sender, whereas the POST data has to be formatted as **name/value pairs** in the request. When the HTTP Receiver / Sender gets a POST request, it behaves like any other Adapter/Receiver combination.

It looks in its **General Container** for matching **Task templates**, instantiates one or more of them into its **Queue** (if a match is found), and fills attributes with values from the POST request.

Let's look at these steps in detail:

- The **Adapter** goes through the templates in its **General Container** and looks for an attribute whose "Type Property" contains the value of the "Type" attribute of the **HTTP Receiver / Sender** (normally "HTTP" but it can be set to anything else).
- If this "Type Property" looks like *HTTP($MATCH_RECURSIVE$,taskmatch)* or *HTTP($MATCH_LOCAL$,taskmatch)*, and if the POST data contains a field called "taskmatch" whose value is identical to the value of the attribute where this "Type Property" is found, then the **Task template** is instantiated into the **Queue**.
- The **Queue** must be a child of the **Adapter** (and therefore a sibling of the HTTP Receiver / Sender). If *$MATCH_LOCAL$* is used, only the **Task template** containing the matching "Type Property" is instantiated. If *$MATCH_RECURSIVE$* is used, this **Task template** and all its child objects are instantiated. (The field name "taskmatch" is of course only used as an example.)
- In the instantiated objects, the Adapter looks for attributes with "Type Properties" which look like HTTP(message) ("HTTP" is again taken from the value of the "Type" attribute of the Receiver). If the POST data contains a field called "message", its value is filled into the attribute with this "Type Property" ("message" is again just an example).
- An attribute with the Type Property HTTP(ConnID) is treated specially. In this attribute the HTTP Receiver / Sender stores an internal ID which makes it possible to identify a TCP/IP connection over which the response should be sent back (if any). This is necessary because the Receiver could talk to many clients at the same time, and if a response should be sent back, it has to know to which of these clients it should be sent. That means that if a setup should be created that requires a response, the response object must contain such a ConnID attribute.

After these steps the Adapter/Receiver is finished with its processing. What happens afterwards is entirely up to the object setup.

11.5 Sending back a response

As mentioned in the previous section, the HTTP Receiver can send back a response to a POST request. The normal Tool/Material method matching is used for this. In the HTTP Receiver / Sender itself (which acts as a sender in this case), there is the Tool method "Send via HTTP" (internal name "Start Sender"). The "Pre constraints" guarantee that this method is only available when the Receiver is started. There must be of course a matching Material method (i.e. with the same [internal] name) to call this method. In addition to this method definition, there must be the following two attributes in the Material object:

MIME Type

The Receiver looks in the Material object for an attribute with a Type Property looking like MIME(application / pdf) (the name of this attribute is not relevant). The contents of this attribute are sent back as the response data. If it is a string attribute, it is interpreted as a file name, and the contents of the file are used. If it is a binary attribute, its contents are used directly. The "Type Property" contains the MIME type to be used for the response data ("application / pdf" in the example). In real applications valid MIME types like "application/pdf" or "image/tiff" would be used of course.

ConnID

This attribute must have the internal name "ConnID", and it must have been filled with a valid connection ID when the POST request was received (i.e. its Type Property must be HTTP(ConnID)). Another prerequisite for a successful response is that the connection to the requesting client (e.g. a web browser) must still be active. If it was broken in the meantime, the response cannot be sent back.

When sending back a response to a POST request, it is absolutely mandatory that the same HTTP Receiver / Sender is used for sending the response which received the request.

Only this HTTP Receiver / Sender knows (by means of the ConnID) to which client the response should be sent.

Before and after sending the response, the HTTP Receiver / Sender also sends events to the Tool and the Material objects to which they can react in their state machine. The event "BeginResponse" is sent before responding, and "EndResponse" is sent afterwards.

If the Receiver should react to them, simply define appropriate **state Transitions** with an "Event Type" of *"Program"* and with "Call Trigger" and "Internal Call Name" set to *"BeginResponse"* or *"EndResponse"*.

Upload of binary data:

In case binary data that might contain control characters (below 0x20) should be uploaded via a form, use MIME type "multipart/form-data" in the form like in the following example:

```
<form name    = "form1"
      encType = "multipart/form-data"
      method  = "post"
      action  = "http://localhost">
  <input name="taskmatch" type="hidden" value="pdf" />
  File to upload:<br>
  <input name="message" size="50" type="file" />
  <input type="submit" value="Submit" />
</form>
```

Escape sequence for quotation marks:

There is no escape sequence required for single and double quotes. To assign a tag attribute with a string containing double quotes, put it between single quotes as depicted below (and vice versa):

```
<input name="linedata" type="hidden" value='<field name="f01">Option 1</field>'
```

11.6 Sample HTTP Receiver

With the knowledge gained from the previous sections the sample setup is easy to understand. The general structure is the same as for all Adapter / Receiver setups.

The Adapter has three child objects:

HTTP Receiver / Sender	transfers data to the Adapter
General Container	contains the Task templates to be instantiated
Queue	Task templates are instantiated into this Queue

11.6.1 HTTP Receiver / Sender

The HTTP Receiver / Sender has two child objects:

- A ROOT folder containing objects which can be requested through the HTTP GET method (by means of a web browser, for example). The sub folder IMAGES contains GIF and JPEG images required for the HTML files of the ROOT folder.
- An ERROR folder containing just the HTML file error.htm. This HTML file is sent back if a GET request fails (e.g. if requesting an unknown object), and it displays an appropriate error message.

11.6.2 General Container

In the General Container two Task templates are located. Each Task template contains an attribute called "MatchCriterion" whose field "Type Property" is set to *HTTP($MATCH_RECURSIVE$,taskmatch)*. If the value of the hidden HTML form field with the name "taskmatch" is set to "pdf" (e.g. <input name="taskmatch" type="hidden" value="pdf">) the Task template "HTTP TASK PDF" along with all its child objects will be instantiated as the attribute "Match Criterion" contains the string "pdf". If the value of the hidden HTML form field with the name "taskmatch" is set to "pol_pdf_brow" (e.g. <input name="taskmatch" type="hidden" value="pol_pdf_brow">) the Task template "Polizze PDF" along with all its child objects will be instantiated as the attribute "Match Criterion" contains the string "pol_pdf_brow".

Task template "Polizze PDF"

The Material "Format Linedata" of both Task templates contain the attribute "Environment variables" whose field "Type Property" is set to *HTTP(message)*, thus the value of the HTML form field "message" sent to the HTTP Receiver / Sender via the HTTP POST method can be later written into an DocEXEC environment variable. When the Task template "HTTP TASK PDF" gets instantiated, the attribute "Environment variables" is used to generate the output. The following example shows a HTML form and a JavaScript used to pass various data to the HTTP Receiver / Sender utilizing the attribute "Environment variables":

```
<script type="text/javascript">
function fix_envar()
{
document.testform.message.value =
"$ENV_LANG=ENG" +
",$ENV_FIRSTNAME=" + document.testform.FIRSTNAME.value +
",$ENV_LASTNAME=" + document.testform.LASTNAME.value +
",$ENV_MALE=" + document.testform.GENDER[0].checked;
return true;
}
</script>

<form method="POST" action="pdfdocument" name="testform"
onSubmit="return fix_envar();">
<input name="message" type="hidden" value="hh" />
<input name="taskmatch" type="hidden" value="pol_pdf_brow" />
First name:
<input type="text" name="FIRSTNAME" size="33" tabindex="1"
value="Valued">
<br>
Last name:
<input type="text" name="LASTNAME" size="33" tabindex="2"
value="Customer">
<br>
Gender:   
<input type="radio" value="MALE" name="GENDER"
tabindex="3" checked>male
<input type="radio" value="FEMALE" name="GENDER"
tabindex="4">female
<input type="submit" value="Submit" name="B1"
tabindex="5">     
<input type="reset" value="Reset" name="B2"
tabindex="6">
</form>
```

Task template "HTTP TASK PDF"

The Data object "Linedata", attached to the Material "Format Linedata" of both Task templates contain the attribute "LineData Name" which also receives the value of the HTML form field "message" via HTTP POST method, as its field "Type Property" is set to *HTTP(message)*. When the Task template "HTTP TASK PDF" gets instantiated, the attribute "LineData Name" is used to generate the output. DocEXEC will process this binary attribute as a Linedata Input File to generate Print lines. The following example shows a HTML form used to pass a text to the HTTP Receiver / Sender:

```
<form name="testform"action="pdfdocument" method="post">
<input type="hidden" name="taskmatch" value="pdf" checked>
The following text will be passed to DocEXEC
(via HTTP Post method):<br>
<textarea name="message" cols="85" rows="4">To acknowledge
what is known as known and what is not known as not known
is knowledge (Confucius).</textarea>
<center>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</center>
</form>
```

The instance created from the Task template "PDF" will contain the PDF data to be sent back as a response.

The Material "Send via HTTP" contains a reference to the data object that contains the finished output file as binary attribute (AFP, PDF, ...). The attribute "ConnID" knows to whom the Response has to be sent, via a reference to the HTTP Receiver / Sender it is possible to send back the response. Because of this, this template has a "PDF Name" attribute with a Type Property of MIME(application/pdf), and an attribute "ConnID" into which the Receiver writes its connection ID. This template also has a special state machine which changes the state of the object from the "Ready" to "Sent via HTTP" when the Receiver has successfully sent the response.

11.6.3 Queue

The Queue contains the Tools for processing the instantiated Task:

- DocEXEC
- PDF converter
- HTTP Receiver / Sender

The HTTP Receiver / Sender in the Queue and the one below the Adapter are references to the same object.

This can be checked by comparing their Object IDs; they are identical. As already mentioned, this must be the case. A different HTTP Receiver / Sender would not know where to send the response.

When the Task template described above is instantiated into the Queue, and the Agent is running, the Agent is able to let the Tool "DocEXEC" process the Material "Format Linedata", thus starting a DocEXEC run.

The provided DOCDEF code will take the contents of the ENV_MESSAGE environment variable and write it into an AFPDS.

When the DocEXEC finishes, the state of the AFPDS object is set to "Ready" which causes the Agent to start the PDF converter. The PDF converter converts the AFPDS into PDF and sets the state of the PDF object to "Ready". As soon as the PDF object is in state "Ready", the Agent drags and drops it onto the HTTP Receiver which sends the created PDF file back to the client. If the client is a web browser with the Adobe Acrobat Reader plugin installed, it will display the PDF.

The Data object "PDF (Binary)", which is the output object in the Material "Print AFP", is also referenced in the Material "Send via HTTP" as input object. As soon as the Data object "PDF (Binary)" changes its state to "Ready" it sends the event "allparents.Goto_Ready(self,x)" to its parents, thus the Material "Send via HTTP" changes its state to "Ready" as well. The Agent is now able to start the tool/material method "Send via HTTP" as the expression "Enginestate==1" in the field "Pre Constraints" of the Material method "Send via HTTP", which is defined in the class "SEND HTTP", evaluates to true. As the "HTTP Receiver/Sender" is in state "Ready" after being started by its Adapter, the expression "Enginestate==1" in the field "Pre Constraints" of the Tool method "Send via HTTP", which is defined in the class "HTTP RECEIVER SENDER", evaluates to true as well.

12.0 PPSClient

PPSClient (ppsclient.exe) is a command line tool which allows to send an ICD file and / or a data file via TCP/IP from a computer without Papyrus objects installed to a computer with Papyrus Objects installed, running a PPSClient Receiver. This Receiver passes the contents of the ICD file in the form of name-value pairs to its Adapter.

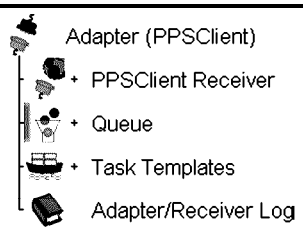


Figure 8: PPSClient Receiver setup

Important:
PPSClient was developed for backward compatibility reasons, it is not recommended to use it in new setups!

12.1 Synopsis

```
ppsclient -h <host> -p <port> send<queue name> <ICD file> [data file]
```

12.2 Options

-h

Specify the hostname of the remote machine where the PPSClient Receiver is running on. You may enter an IP address or a symbolic hostname.
Default: localhost

-p

Specify the port on which the PPSClient Receiver is listening for incoming requests.
Default: 2006

send

Specify the following parameters:

Queue name

Specify a string which the PPSClient Receiver will pass to the Adapter as parameter "QueuemanagerQueueName".

ICD file

Specify the ICD file which should be sent to the PPSClient Receiver. The name of this file will be passed to the Adapter as parameter "DispatchFileName", its extension as parameter "DispatchFileExt". If no ICD file but only a data file should be sent, a dash must be specified as placeholder.

Data file

Optionally, a data file may be specified which is sent to the PPSClient Receiver. At the Receiver side, this file will be stored as temporary file whose name will be passed to the Adapter as parameter "PPMFInputName".

13.0 Output of externally generated print data streams

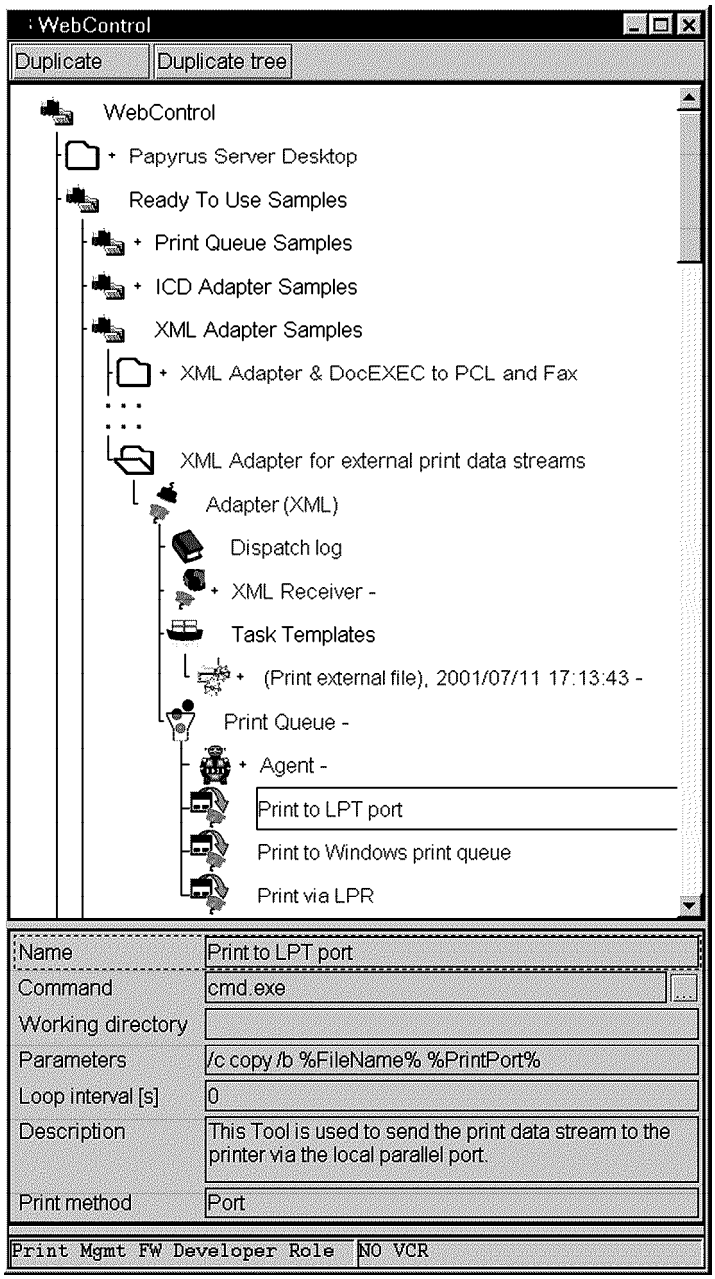
Papyrus WebControl does not only allow to control ISIS processes and print tasks but also to pick up externally generated print data streams like PCL, PostScript, PDF, etc. and route them through the Papyrus system to the target printer. This enables fully integrated control and monitoring features for all production steps within one system used by the operators. To output print data streams generated by non ISIS applications (e.g. PDF generated by Adobe Acrobat or PCL generated by an external application) under Papyrus WebControl or WebRepository two steps are mainly required:

(a) Interface to the external data stream

(b) Define parameters for the route through step (printer name, IP address, ...)

The interface to external data can be any Adapter setup provided by the Papyrus objects System like XML, ICD, HTTP, MQ, SAP, etc. The Receiver picks up the external data and passes it to the Adapter, which then searches for a matching Task template containing the route through steps. An instance of the class "EXTERNAL APPLICATION" can be used as a Tool for the 'virtual printer' to be addressed to send the required information found in the Material (instantiated by the Adapter from the Task templates folder) to an application that generates a print out (e.g. binary copy to LPT, copy to IP address, routing via LPR, etc.).

A Ready To Use Sample containing three differently set up Tools for printing can be found in the Library folder "Library / WebControl / Ready To Use Samples / XML Adapter Samples / XML Adapter for external print data streams":



Example:

To utilize the class "EXTERNAL APPLICATION", create a template and specify path and filename of the executable via the attribute "Command" and options for this executable via the attribute "Parameters". In the above example the Windows Command Processor cmd.exe is started to run the windows command `COPY` which is used to send the data stream to a parallel printer port. Finally a Task containing a Material with attributes for the printer connection settings and path and filename of the print data stream has to be defined. The Material template in this example is derived from the class "CALL EXTERNAL" and contains the attributes "Printer method", "Printer port", "Print queue", "LPR Host", "LPR printer", and "File to print". Match criterions like *XML(Printermethod)* are defined in the field "Type property" of each of these attributes, thus the Adapter will fill the attributes with values from the XML file.

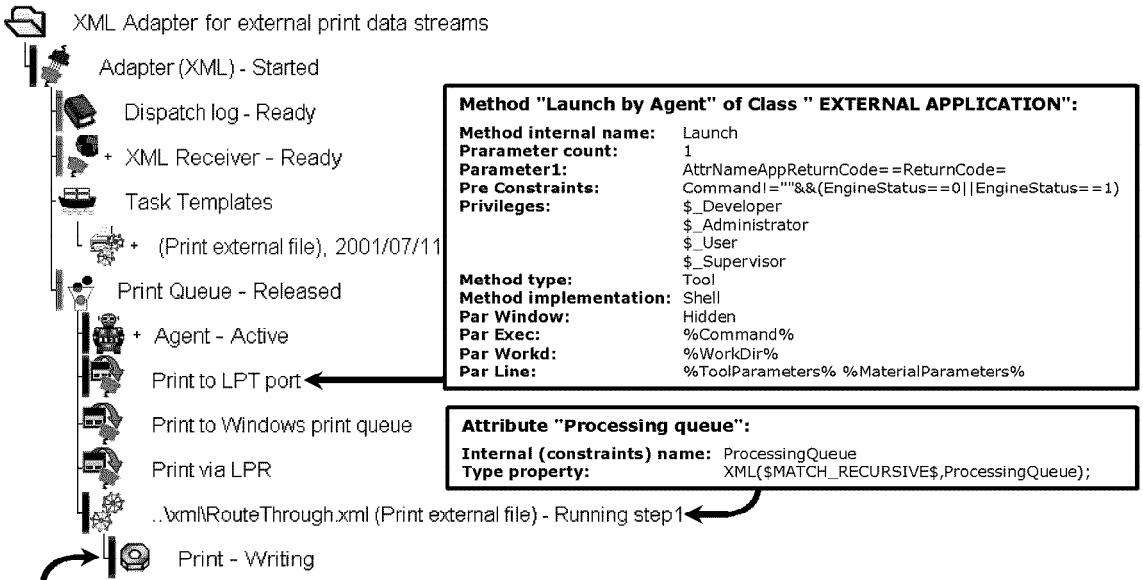
By dropping an XML file, which contains the printer connection settings and the filename of the print data stream to be output, into a trigger directory monitored by an XML Receiver, the print data stream can be printed via Papyrus WebControl.

The XML file in this example, "RouteThrough.xml", contains the tag <*ProcessingQueue*> which is used by the Adapter to instantiate the appropriate Task template via the match criterion *XML($MATCH_RECURSIVE$,ProcessingQueue)* in the field "Type property" of the Task attribute "Processing Queue". Path and filename of the input file are specified via the tag <*TargetName*> and mapped into the attribute "File name" of the Material "Print" via the match criterion *XML(FileName)*. The print method, port and queue as well as the LPR definitions are specified via the XML tags <*Printermethod*>, <*PrinterPort*>, <*PrintQueue*>, <*LPRHost*>, and <*LPRPrinter*>. They are mapped into their respective attributes via match criterions like *XML(PrinterPort)*.

XML file RouteThrough.xml:

```
<XML>
  <ROUTETHROUGH>

    <DESCRIPTION>This is sample XML job for demonstrating the use of Papyrus Objects
    to route existing print data streams to a printer.</DESCRIPTION>
    <PROCESSINGQUEUE>ProcessingQueue</PROCESSINGQUEUE>

    <COMPLETEDQUEUE>CompletedQueue</COMPLETEDQUEUE>

    <ERRORQUEUE>ErrorQueue</ERRORQUEUE>

    <FileName>E:\isis\data\ASCHotel.asc</FileName>

    <Printmethod>Queue</Printmethod>

    <PrinterPort>LPT1</PrinterPort>

    <PrintQueue>\\printbt3og\IBMInfop</PrintQueue>

    <LPRHost>10.1.5.5</LPRHost>

    <LPRPrinter>35</LPRPrinter>

  </ROUTETHROUGH>
</XML>
```

The following Illustration shows a typical setup for putting out print data streams of external sources and how the XML file is mapped into certain attributes.



In addition the state of the Material is changed based on the return code of the process outputting the external file. The return code is also written to the attribute "ReturnCode" of the Material "Print". A *Send Event* is then used to change the state of the Task as well (e.g. to automatically move the finished Task into a Completed Queue).

As soon as an suitable XML file is dropped into the Trigger directory of the XML Receiver, a Task template is instantiated by the Adapter and the corresponding attributes are filled with the values in the XML file. The Material and the Tool both contain the attribute "Print method", which is used by the Agent to select the appropriate Tool for the Material (e.g. "Print via LPR").

14.0 MQ Receiver and MQ Sender

The MQ Receiver and the MQ Sender (MQ-Send) are used to receive message from and send messages to IBM WebSphere MQ (IBM MQSeries). The MQ modules ("MQ Receiver" and "MQ-Send") have been successfully tested for local and remote Queues under MQ version 5.2. There is no guarantee that it will work properly with other versions as long as the MQ vendor's release notes do not tell that it is backward compatible with V5.2. We recommend to perform tests first. If a guarantee for another version is required, please contact the ISIS Marketing Department. Ready-To-Use samples for MQ Receiver and MQ Sender can be found in the Library folder "Domain Controller / Library / Communication via MQ".

14.1 MQ Receiver

MQ Receiver uses the MQClient to establish a connection to IBM WebSphere MQ (IBM MQSeries) and forwards the incoming messages to the Adapter. The Adapter looks for a matching Task template in its General Container to instantiate it into its Queue and populates the attributes of the Task and its child objects with the data transmitted by the MQ Receiver. MQ Receiver is able to process segmented and unsegmented messages and automatically recognizes the difference via the message header (whether the message header contains the message descriptor "MDE" or not). The size of unsegmented messages is arbitrary and depends on the size defined for the MQ-Queue. Segmented messages, which are parts of messages bigger than the size defined for the MQ-Queue (e.g. binary content split into smaller parts when put into the MQ-Queue), are automatically merged by the MQ Receiver. The MQ Receiver can be used in a workflow in combination with the MQ Sender to also handle incoming reply messages. While the MQ Sender uses only one connection per process, the MQ Receiver process establishes always 2 MQ connections. One for listening to the MQ-Queue and the other one to extract the message and send it to the Adapter.

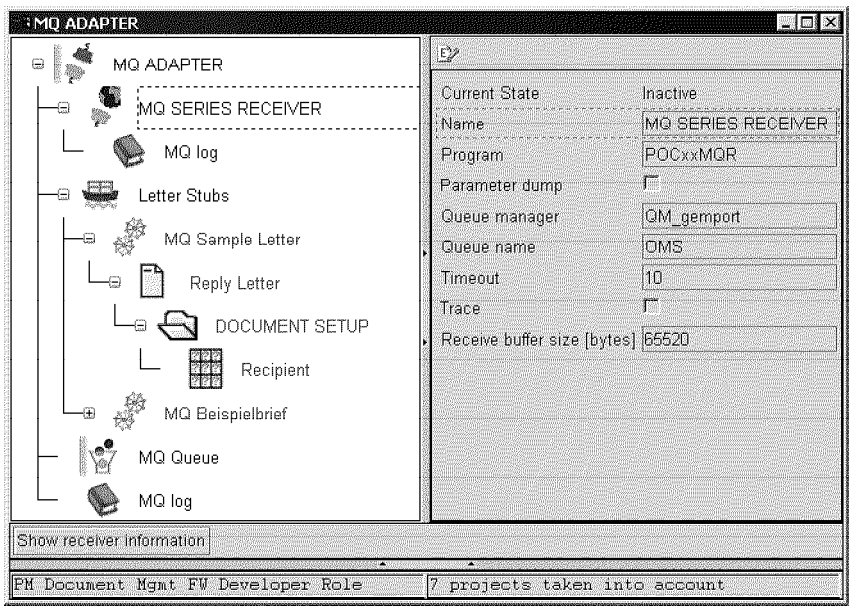


Figure 9: Typical setup of a MQ Series Receiver

14.1.1 States

No state machine defined, thus the ones defined in the base class "ADAPTER RECEIVER" are also valid for this class.

14.1.2 Methods

Show receiver information	Outputs general information about processes performed by the MQ Receiver in the message panel including all attributes.

14.1.3 Attributes

Program	Executable behind the MQ Receiver. As the MQ Receiver is available for various platforms, the identifier POCxxMQR is defined, thus the system will retrieve path and filename from the appropriate Program object.
Parameter dump	If this option is selected, the start parameters are dumped.
Queue manager	Queue manager defined on MQ server to which the MQ Receiver connects.
Queue name	MQ-Queue on which the MQ Receiver is listening.
Timeout	The MQ Receiver waits for an answer by IBM WebSphere MQ (IBM MQSeries) until the specified timeout is reached (default value = 10 seconds).
Trace	If this option is selected, additional information is output in the message panel.
Receive buffer size [bytes]	Buffer size for received messages. Set to appropriate size depending on the maximum message length that has to be processed by the MQ Receiver (default = 65520 bytes).

14.1.4 Setup of MQ Receiver

14.1.4.1 Match criterion for the instantiation

To set up an MQ Receiver, the sample in the Library folder "Library / Adapters / Adapter (MQ)" can be extended by one or more Task templates.

To enable the Adapter to instantiate the appropriate Task template based on data of the incoming message, one attribute of the Task requires a match critierion in the field "Type property" with the following syntax:

```
AdapterType(Keyword,Position,Length,Codepage)
```

The placeholder **AdapterType** has to be exchanged by the string defined in the attribute "Type" of the Adapter. Via the attribute "Type" it is possible to identify an Adapter, thus a match criterion starting with the string "MQ" for instance only applies to the Adapter containing the string "MQ" in its attribute "Type". The placeholder **Keyword** has to be exchanged by $MATCH_RECURSIVE$, to let the Adapter instantiate the Task template along with all its child objects, or $MATCH_LOCAL$, to let the Adapter instantiate the Task template without its children. For the placeholder **Position** enter the offset in the incoming message at which the string to be compared should start and for placeholder **Length** enter the length of the string to be compared. The placeholder **Codepage** has to be exchanged by the codepage used in the MQ message, thus it will be converted to the codepage of the Node (e.g. an MQ message with codepage 273 comming from a host).

Example:

The match criterion MQ($MATCH_RECURSIVE$,8,6) for instance, defined in the field "Type property" of an additional attribute of the Task, would be used by the Adapter to check if the 6 digits string at position 8 of the received data is equal to the contents of the attribute.

14.1.4.2 Match criteria to populate attributes with the received header data

To fill the attributes of the Task and its child objects with the header data received by IBM WebSphere MQ (IBM MQSeries), use match criteria in the field "Type property" of the attribute with the following syntax:

```
AdapterType(MessageDescriptor,Position,Length,Codepage)
```

The **Message Descriptor** defines which header type the received MQ data is supposed to have, whereas the following header types can be defined in the match criteria:

MQMDE (default)

The data is sent as BLOB and if there is a header, it is put before the data. The offset of the data has to be known to separate header from data via mach criteria (e.g. match criterion MQ(MQMDE, 1, 6) to retrieve six digits header).

Example:

In the example below attributes has been defined with the following match criteria in the field "Type property" to retrieve the header (starting at position 1 with a length of 10 digits). As MQMDE is the default header type, it can be omitted in the match criterion.

```
MQ(MQMDE,1,10) or MQ(1,10) ... to receive the header data
```

MQRFH

The data is sent in a BLOB and the header will be send in a separate memory array, which can be used by the Adapter to locate the header data. Header data can be analyzed in the Adapter as the header type information will be passed like Param="value", where Param will be 'MQRFH'. An attribute with the match criterion MQ(MQHRF) in the field "Type property" can be used to receive the header data.

Example:

In the example below an attribute has been defined with the following match criterion in the field "Type property" to retrieve the header.

```
MQ(MQHRF) ... to receive the header data.
```

MQRFH2

The data is sent in a BLOB and the headers will be send in a separate memory arrays, which can be used by the Adapter to locate the header data. Header data can be analyzed in the Adapter as the header type information will be passed like Param="value", where Param will be 'MQHRF2, nnn'. attributes with the match criterion MQ(MQHRF2, nnn) in the field "Type property" can be used to receive the header data.

Example:

In the example below three attributes have been defined with the following match criteria in the field "Type property" to retrieve three headers.

```
MQ(MQHRF2, 000) ... to receive the first header data
MQ(MQHRF2, 001) ... to receive the second header data
MQ(MQHRF2, 002) ... to receive the third header data
```

14.1.4.3 Match criteria to populate attributes with the received data

To fill the attributes of the Task and its child objects with the data received by IBM WebSphere MQ (IBM MQSeries), use match criteria in the field "Type property" with the following syntax:

```
AdapterType(Position,Length,Codepage)
```

The number of attributes and the offsets and lengths of the datagrams depends on what is defined in IBM WebSphere MQ (IBM MQSeries).

Example 1:

In the example below three attributes have been defined with the following match criteria in the field "Type property" to retrieve the header and three datagrams with a length of 30 characters each and starting at offset 10, 30, and 60. Since the data comes from a host the correct codepage (in this case 273) has been defined.

```
MQ(MQHRF)
MQ(10,30,273)
MQ(30,30,273)
MQ(60,30,273)
```

Example 2:

If the data structure sent by IBM WebSphere MQ (IBM MQSeries) changes, the attributes will be populated with wrong values (e.g. attributes "Name" and "ZIP"). In such a case the match criteria used to populate the attributes have to be adjusted to the new data structure (different lengths affecting offsets) like in the following example:

Old MQ string **"OLD Miller123456"** with only 6 digits reserved for the name and new MQ string **"NEW Miller 123456"** with 10 digits for the name affecting the offset of the ZIP code.

If both, the old and the new data structure, shall be processed, two Task templates are required where the attribute with the match criterion for instantiation (e.g. defined in attribute "Match") is able to distinguish the data structures by checking a characteristic string (e.g. first 3 characters in example above).

Task template settings for OLD:

Match = MQ($MATCH_RECURSIVE$,1,3) = "OLD"

Name = MQ(8,**6**) = "Miller"

ZIP = MQ(**14**,6) = "123456"

Task template settings for NEW:

Match = MQ($MATCH_RECURSIVE$,1,3) = "NEW"

Name = MQ(8,**10**) = "Miller "

ZIP = MQ(**18**,6) = "123456"

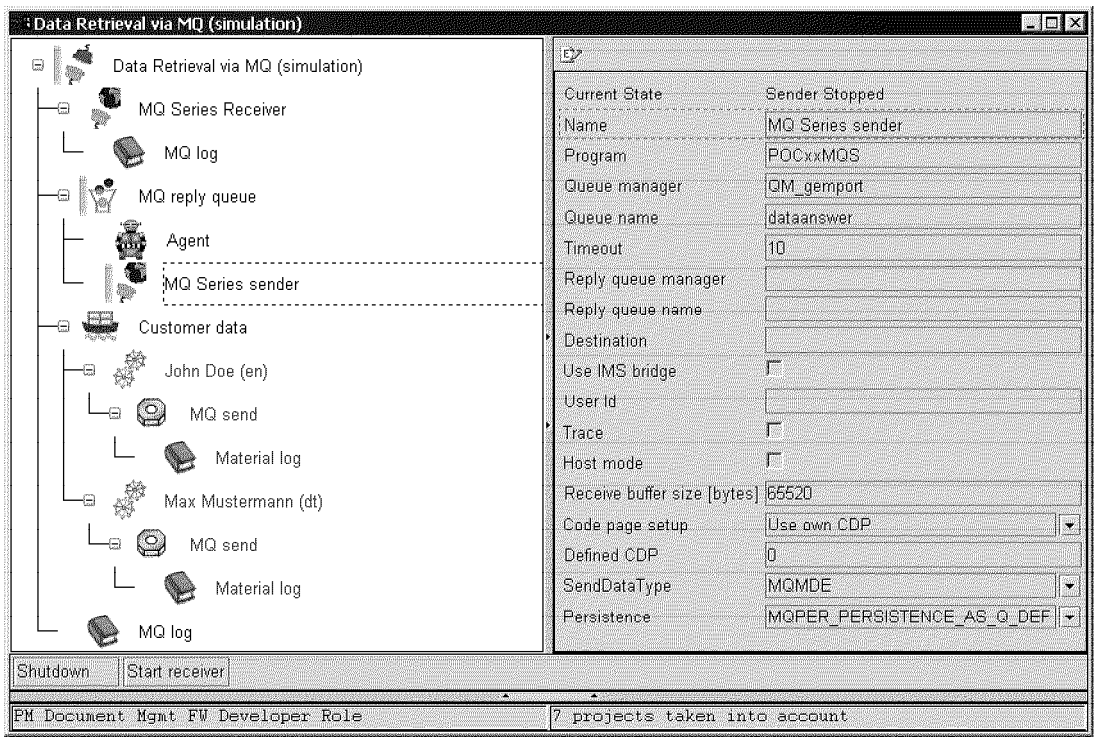
14.1.4.4 Match criteria to populate attributes with the Message ID and the Correlation ID

To fill the attributes of the Task and its child objects with the message and the correlation ID received by IBM WebSphere MQ (IBM MQSeries), use match criteria in the field "Type property" with the following syntax:

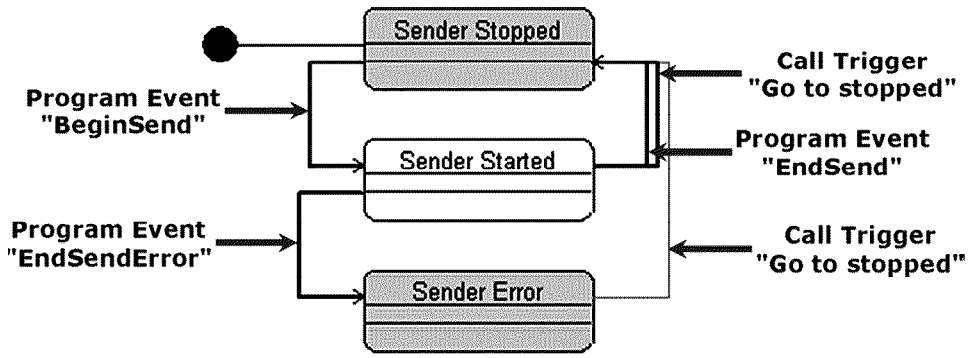
```
AdapterType(CORREL_ID)
AdapterType(MESSAGE_ID)
```

14.2 MQ Sender

MQ Sender (also known as "MQ-Send") is a component of Papyrus Objects which acts as a Tool. It can send and optionally receive (reply-)messages via IBM WebSphere MQ (IBM MQSeries). Its purpose is not to receive messages to process them like the MQ Receiver. The method "MQ Series send" (defined in the class "MQ SERIES SENDER") is not only able to send an outbound message from Papyrus Objects to a MQ-Queue but can also handle one or more incoming reply messages. An outbound message is called a *Request*, an (corresponding) inbound message is called the *Answer*. IBM WebSphere MQ (IBM MQSeries) is comparable to a postal service: it just transports messages not caring about the content. This means that the applications which are communicating are responsible for the content. When MQ Sender is called the program sends data to the MQ Queue defined in the MQ Sender. The Material "SEND MQ", used to send the data is then in a "processing" state. MQ Sender is listening to the reply queue of IBM WebSphere MQ (IBM MQSeries). When the reply queue receives an answer the Material "MQ send" changes to a "finished" state. While the MQ Sender uses only one connection per process, the MQ Receiver process establishes always 2 MQ connections. One for listening to the MQ-Queue and the other one to extract the message and send it to the Adapter.



14.2.1 States



14.2.2 Methods

MQ Series send (Start)	Tool method used in combination with the matching Material method "MQ Series send (Start)" defined in the Material class "SEND MQ" to send MQ messages.
Shutdown (Shutdown)	Action method to shut down the MQ Sender.
Print pool (PrintPool)	Define this additional method to use MQ Sender in a special mode, which allows to send messages formatted in a way that Papyrus Host is able to process them with its MQ Interface. Details are described in chapter "Papyrus Host MQSeries Interface" of the manual *"Papyrus Host, Installation and Development Guide"*. If MQ Sender is used in this Print Pool mode, it is important to fill the attribute "ReplyQueue". Otherwise no reply will be generated. Papyrus Host looks for the name of the ReplyQueue in the message and uses this information, to know where to send the answer. MQ Sender inserts this piece of information automatically in the request.
Start receiver (WakeUp)	Actively start MQ Sender without the requirement to send an mq message to be able to receive messages.

The methods "Delete reference" and "Move reference" have been overloaded from the MetaClass to limit the access to developers and administrators.

14.2.3 Attributes

Program	Executable behind MQ Sender. As MQ Sender is available for various platforms, the identifier POCxxMQS is defined, thus the system will retrieve path and filename from the appropriate Program object.
Queue manager	Queue manager defined on MQ server to which the MQ Sender connects.
Queue name	MQ-Queue on which the MQ Sender is listening.
Timeout	If the user tries to shut down the MQ Receiver, it waits for reply messages in the reply queue for the specified time and uses the Correlation ID to send back the answer.
Reply queue manager	Reply queue manager defined on MQ server to which the MQ Sender connects.
Reply queue name	Reply queue on which the MQ Sender is listening for reply messages.
Destination	This attribute can be used to define a match criterion of type MUST_MATCH or CAN_MATCH, thus the Agent will only start the Tool / Material method "MQ Series Send" between MQ Sender and Material "SEND MQ" of the incoming MQ Task if the corresponding attributes contain the same value (e.g. three MQ Sender in the MQ Reply Queue for one incoming MQ Task).
Use IMS bridge	If this option is selected the MQ Sender adds a header in IMS format to the MQ message to send it in a host compatible way.
User Id	By default the login name of the system is used, which is stored in the header of the MQ message. To use a different login name, specify it via this attribute.
Trace	If this option is selected, additional information is output in the message panel.
Host mode	If this option is selected, a trailer record is added at the end of segmented MQ messages, thus the host will be able to concatenate the message parts.
Receive buffer size [bytes]	Buffer size for received messages. Set to appropriate size depending on the maximum message length that has to be processed by the MQ Sender (default = 65520 bytes).
Code page setup	The following options are available to define whether the codepage of the MQ server, the own codepage, or a codepage specified via attribute "Defined CDP" should be used for the message to be sent: Translate to CDP of Destination Use own CDP Send with dedicated CDP

Defined CDP

Number of code page if option "Send with dedicated CDP" is selected in attribute "Code page setup".

Send data type

Message Descriptor defining the header type for sending or receiving MQ data via MQ Receiver or MQ Sender. The following header types can be selected:

MQMDE (default)
This header type is used to send binary data to MQ using the keyword MQSendData() in the field "Type property" of an attribute and optionally a header via keyword MQSendHeader() in the field "Type property" of another attribute.
MQRFH
This header type (also known as MQHRF2) is used to send exactly one header in addition to the data. Header and data can be divided on the receiving part as this header type contains information about its length.
MQRFH2
This header type (also known as MQHRF) is similar to header type "MQRFH", but an arbitrary number of headers can be send via the same number of attributes with keyword MQSendHeader() defined in their field "Type property". Headers and data can be divided on the receiving part as this header type contains information about its length. This header type is required if MQ Sender is used as interface to JMS (Java Message Service), which usually receives three "folders" (=headers) containing XML structures and one data packet.

Example:
In the example below four attributes have been defined and they wil be combined by MQ Sender in the order they have been defined to build the MQ message. For header type MQRFH2 the first three attributes have the keyword MQSendHead() defined in their field "Type property" and the following contents:
1st attribute = <mcd><Msd>jms_text</Msd></mcd>
2nd attribute =
<jms><jmsprop01>text01</jmsprop01></jms>
3rd attribute =
<user><userprop01>utext01</userprop01></user>

The last attribute has the keyword MQSendData() defined in its field "Type property" to carry the data.

Persistence	If an MQ-Queue uses persistence, its messages will be recovered after a restart of the MQ server, otherwise they would get lost. The following options are used to override the persistence definition of the MQ-Queue. **MQPER_PERSISTENCE_AS_Q_DEF** (default) Persistence definition of the MQ-Queue will be used for the sent message. **MQPER_PERSISTENT** Use persistence for the sent message. **MQPER_NOT_PERSISTENT** Do not use persistence for the sent message.
Retry attempts	Number of retries to open a queue (in case of conflicting settings).
Retry interval (s)	Interval in seconds between retries to open a queue.

All attributes not listed above are deactivated and require no administration.

14.2.4 Setup of MQ Sender

There following main points have to be considered:

- *Structure of the content*
  Only a string (sequence of "readable" chars) is transported, if this string consists of several fields (in a sense of a structure in C and C++ or a COBOL-Record), then somebody should be able to describe these fields with a fixed offset and a fixed length.
  With the MQ Sender it is not only possible to send AFPDS via PHP but also any other kind of binary file (e.g. an AFPDS via the MQ-Interface of Papyrus Host into the Print Pool). In this case the outbound message is not constructed with fixed length fields. This processing mode will be described later.

- *Platform specific information (code pages)*
  The inbound message may be sent away by a different platform, which may be characterized by a code page. As IBM WebSphere MQ (IBM MQSeries) usually does no conversion, it is necessary to convert the message string into the right code page, which MQ Sender and Papyrus Objects are using internally (e.g. 1252 on Windows).

- *Codepage*
  Outbound messages are always sent in code page 1252.

- *Linking an inbound message to a previously send outbound message*
  This linking is done by a means of IBM WebSphere MQ (IBM MQSeries), the Correlation ID, and the Message ID. If these IDs (in a reply message) are matching to these of a sent-away-message, the reply is taken as the answer to the message before.

14.2.5 Definition of outbound messages

MQ Sender constructs requests out of values of attributes. These attributes belong to the Material "SEND MQ", which is actually processed by MQ Sender. MQ Sender just concatenates these values in a well defined order. This order is defined by using the expression *MQSend(n)* for unsegmented messages and *MQSendData(n)* for segmented messages in the field "Type property", where *n* is an integer greater or equal to 1. Do not use a mixture of *MQSend(n)* and *MQSendData(n)* within one call!

To send one or more headers, a message descriptor has to be selected in attribute "Send data type" of the MQ Sender (see table above).

If data is send with the expression MQSendData(n) in the field "Type property" of the respective attribute and either the Tool "MQ Series Sender" or the Material "MQ SEND" contains an attribute with the Internal name "MQSendJobID", it will be filled at send time with a unique JobID. This JobID will also be sent at the begin of the MQ message data as an ISIS-specific header within the MQ message. The string which will be added is `REQUESTJOBID = <JobID>`. If no attribute with the Internal name "MQSendJobID" can be found, the JobID will be omitted.

Example:

If the request should be made of attributes REQ1 and REQ2, then the Type Property of REQ1 has to be MQSend(1) and the Type Property of REQ2 has to be MQSend(2). The complete request will be a concatenation of the attribute values. If REQ1="Request" and REQ2="Inquiry" then the request would be "RequestInquiry".

If the value of an attribute has to be placed at an specific offset (e.g. the value of REQ2 at offset 10), then a third DUMMY-attribute has to be defined with 3 filling characters (e.g. "???" as value), its Type Property would be MQSend(2). The Type Property of REQ2 has to be changed to MQSend(3). This leads to a request like "Request???Inquiry", with the value of REQ2 at offset 10.

14.2.6 Analysis of inbound messages

Once MQ Sender is started, it is able to receive answers. It is possible to extract parts of these answers into attributes of instances, therefore the following prerequisites have to be fulfilled:

- The Material "SEND MQ" requires an attribute called MQType (MQType). The value of such an attribute defines the Prefix of the Type Property, with which the extraction of parts of the answer into one or more attributes can be done. If no attribute with this name is given, MQ Sender will use "MQ" as a default.
- The Type Properties have exactly the same syntax as those which are used in the templates in the MQ Receiver scenario:
  <prefix>(offset,length,codepage). These attributes do not necessarily have to be in the Material "SEND MQ", which was used do define the request. It is also possible to populate attributes of the child objects of the Material as class "SEND MQ" has the child reference "Child (Child)" defined with option "Search attributes" checked. It is possible to use the same parts of the answer in different attributes.
- If the Correlation-ID and the Message-ID in a reply-message match those of a send-away-message, the reply is taken as the answer to the message before.

Example:

The value of the MQType attribute is "MQ". The sender sends with code page 1252, the answer looks like "aaaaabbbbbccccc". The field at offset 0, length 5 has to be filled into an attribute REQ1_Answer and the content at offset 8, length 6 belongs into attribute REQ2_Answer, then the Type Properties of REQ1_Answer look like MQ(0,5,1252) and those for REQ2_Answer are MQ(8,6,1252). After receiving the answer the value of REQ1_Answer will then be "aaaaa" and the value of REQ_Answer is "bbbccc". The rest of the message is ignored.

If MQ Sender is not active (i.e. mqsend.exe is not running) answers will be ignored. They will be processed after the next start of MQ Sender, assuming that "nobody else" is reading (and deleting) these messages from the ReplyQueue.

14.2.7 Definition of the Material MQSendToPrintPool

Instances of the class MQSendToPrintPool represent the request which transports an AFPDS via Papyrus Host into the Print Pool.

All following attributes have the Type Property MQSendHead(1), the (internal) names of these attributes are ignored, MQ Sender identifies the necessary attributes by the Type Properties. The content of each attributes is always *<Keyword>=<Value>*.

```
Head 1 (MQA_JOBNAME): REQUESTJOBNAME=MYJOB
Head 2 (MQA_JOBID): REQUESTJOBID=JOB00
Head 3 (MQA_USERNAME): REQUESTUSERNAME=USER
Head 4 (MQA_PROGRAMMER): REQUESTPROGRAMMER=ISIS_PROGRAMMER
Head 5 (MQA_CLASS): CLASS=0
Head 6 (MQA_NOTIFY): NOTIFY=TEST
```

The values for the keyword are defined at the time of the installation of Papyrus Host:

- MQSendJobId (MQSendJobId)
  This attribute is filled when MQ Sender reads the answer. It's purpose is to show a relationship between the local process in Papyrus Objects an the activities on the mainframe and can be used for debugging. MQ Sender does not know about this attribute via a Type Property but uses the hardcoded Internal name "MQSendJobId" instead.
- MQType (MQType)
  This attribute defines the Prefix used in Type Properties for the extraction of parts of the answer into one or more attributes (the default value is "MQPP").
- RC (RC)
  This attribute contains the return code of the job after receiving the message, which Papyrus Host was processing. The return code is extracted with the Type Property MQPP(0,4,273).
- MessageDATE (MessageDATE)
  This attribute contains the time stamp of the job, the value is extracted with the Type Property MQPP(7,26,273) from the message.
- MessageID (MessageID)
  This attribute contains the ID of the job's message, the ID is extracted with MQPP(33,8,273).
- MessageTXT (MessageTXT)
  This attribute contains a textual description of the job's result, the text is extracted with MQPP(42,90,273). The Material has to be used with the Data object AFPDS which contains the AFP as binary attribute or file attribute for the check-in into the Print Pool.
- The Type Property of the AFPDS attribute have to contain MQSendData(1). This Type Property is used to assign this AFPDS to the corresponding head information, which is constructed by all attributes which have the Type Property MQSendHead(1).
- With the Type Properties MQSendData(n) and MQSendHead(n), where n is an integer greater or equal to 1 used as an identifier for the "file", it is possible to send more then one document in separated AFPDS into the Print Pool. MQSendData(n) contains the "file" to be send, MQSendHead(n) contains "header information" that corresponds to this "file".
- The return value received in the reply queue is handled as a BLOB, which can be made visible via an attribute that contains the Type Property "MQ(offset,length,codepage)".

14.3 Material "MQ send"

The MQ Material is used to store the data to be sent by MQ Sender and to store the data received by MQ Sender.

14.3.1 states

Empty	Received data can be stored in the MQ Material by the MQ Sender.
Ready	Data stored in the MQ Material can be sent by MQ Sender.
Sending	Data stored in the MQ Material is currently being sent by the MQ Sender.
Receiving	Received data is currently being stored in the MQ Material by the MQ Sender.
Final	All tasks of the MQ Material have been successfully performed.
Sending error	During the sending of data an error occured.
Waiting	This is an intermediate state between "Sending" and "Receiving". After the data has been sent by MQ Sender, the MQ Material is waiting for incoming data forwarded by the MQ Sender.
Receiving error	During the receiving of data an error occured.

14.3.2 Methods

Resend	Changes back to the initial state "Ready" defined in system attribute "Initial state", to be able to send the data again.
MQ Series send	Material method used in combination with the matching Tool method defined in the MQ Sender to send the data.

14.3.3 Attributes

Correlation ID	The Correlation ID sent to IBM WebSphere MQ (IBM MQSeries), which is compared with the Correlation ID in the reply message to write back the data in the appropriate Material. If no Correlation ID has been specified, the Object ID will be used for it.
Message ID	The Message ID sent to IBM WebSphere MQ (IBM MQSeries). If no Message ID has been specified, the Object ID will be used for it.
MQType	Defines the prefix of match criteria used in the field "Type Property", with which the extraction of parts of the answer into one or more attributes can be done. If no MQType has been specified, MQ Sender will use "MQ" as a default.

All attributes not listed above have been overloaded from the base class "MATERIAL" to make them invisible by unchecking option "Visible for edit".

14.4 Using Client Connection Channels under MQ

Using the MQServer variable to access queue manager, only one queue manager can be addressed by one Papyrus Objects Kernel and if the listener port of the queue manager changes it is required to restart the Papyrus Objects Kernel with the new environment variable. These limitations do not exist if client connection channels are used.

As IBM WebSphere MQ (IBM MQSeries) is a third party application, the following description how connection channels can be set up is only an example and might change. Please refer to the current documentation provided by IBM. ISIS Information Systems assumes no liability or responsibility for any loss or damage arising directly or indirectly from using third party software.

14.4.1 Definition of a Client Connection Channel

Starting from the default queue manager, first add the desired client connection:

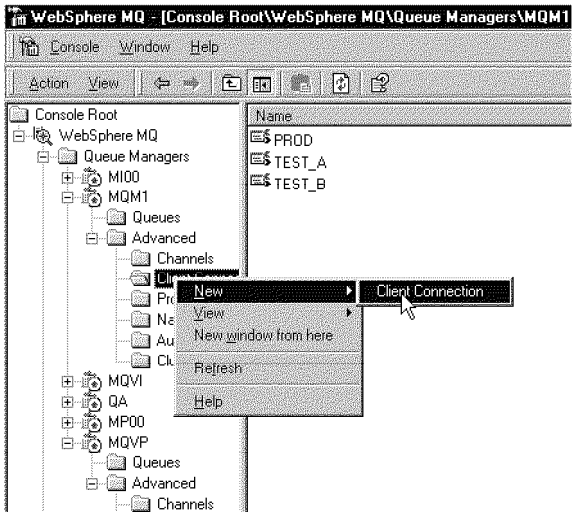
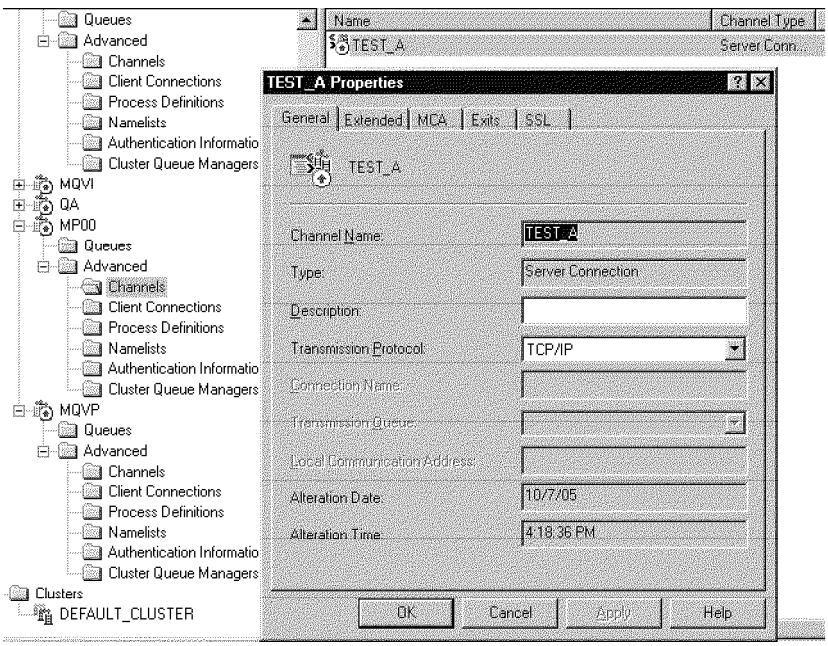


Figure 10:

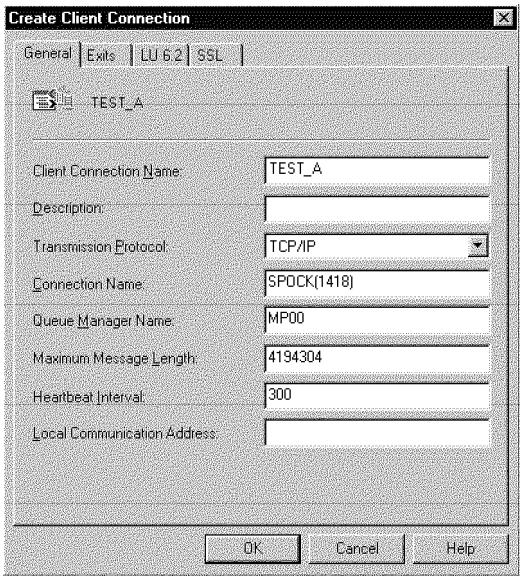
At least the following parameters have to be added:

- Name of the client connection, which has to be the same as the name of the server connection channel on the desired queue manager as depicted below:



- Connection Name, which is the name or IP address of the MQServer including the listener Port address (if different to the default port 1414)

- Name of the Queue Manager:



After confirming the input the client channel table file under the queue manager where the connection is created is updated. With this procedure different connections to various queue managers can be defined.

The channel table file is stored in the program directory of MQServer (e.g. C:\Program Files\IBM\WebSphere MQ\Qmgrs\MQM1\@ipcc\AMQCLCHL.TAB) and has now to been copied to the client machine.

To use the client connection channels instead of the MQServer variable, the following lines have to be added to the start script of the Papyrus Objects Kernel:

```
set MQSERVER=
set MQCHLLIB=<path to channel table file>
set MQCHLTAB=AMQCLCHL.TAB
```

If the channel table file has been changed it is only required to restart the MQ Adapter.

15.0 POP3 Receiver

The POP3 Receiver acts as a POP3 client (Bridge type DLL receives e-mails via default port 110) and forwards the fetched e-mails as name/value pairs to the Adapter. , which instantiates the E-Mail template derived from class "EMAIL" and maps the media types of the received e-mail in separate Data objects below. One or more e-mail account can be defined via the Mailbox Configuration object derived from class "POP3 MAILBOX". The string "POP3" in the field "Type" of the Adapter is used for the match criterions.

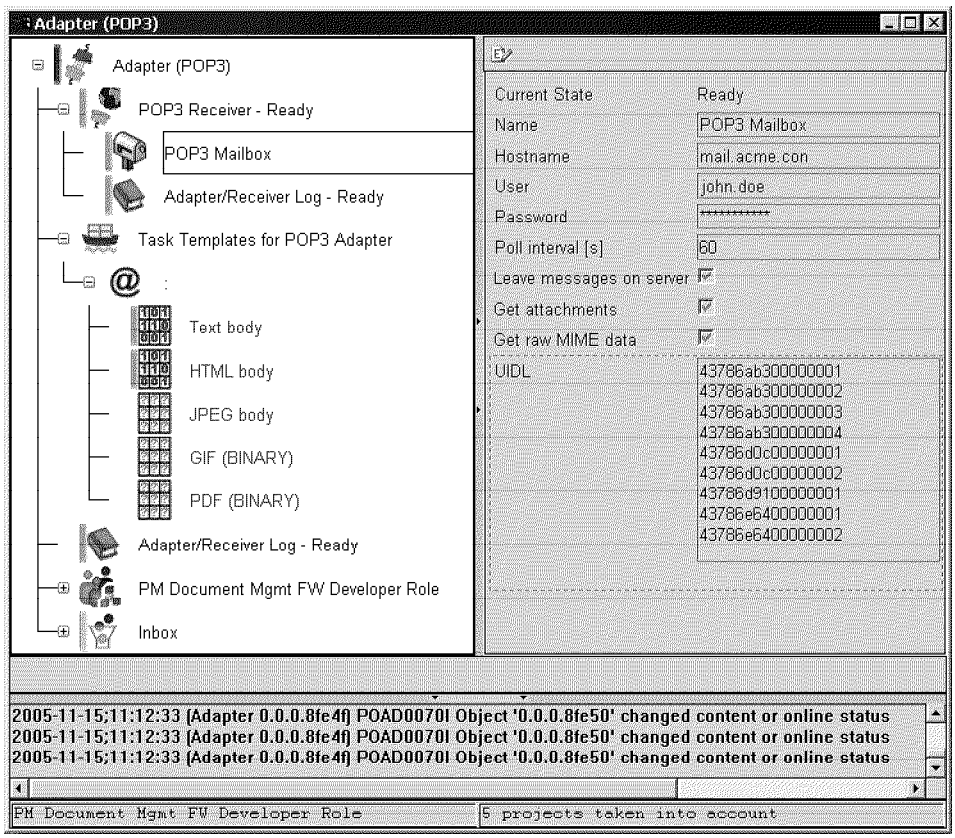


Figure 11: Mailbox configuration object "POP3 MAILBOX"

15.1 States

As there is no state machine defined, the one defined in the base class is taken.

15.2 Attributes

Program Executable behind the POP3 Receiver. As the POP3 Receiver is available for various platforms, the identifier POCxxPOP is defined, thus the system will retrieve path and filename from the appropriate Program object.

15.3 Methods

Check now Via this method it is possible to fetch e-mails immediately without waiting for the next interval defined in attribute "Poll interval [s]" of the Mailbox Configuration object. This method is available after the POP3 Receiver has been automatically started by the Adapter as the Pre constraint is set to "EngineStatus==1".

15.4 Data transmitted to the Adapter

The POP3 Receiver sends a dummy name/value pair **Matchme="Message"** which allows the Adapter to instantiate a template.

The section below lists all message headers recognized by the POP3 Receiver and forwarded to the Adapter. The Adapter will then fill the attributes of the Task and its child objects if it finds an appropriate match criterion in the field "Type Property".

Note: X-lines are nonstandard lines that had been added by many clients and servers, but are actually not part of any official standard. Therefore they are not mandatory any arbitrary X-line can be defined by the user.

Headers containing text only:

POP3 Header	Internal Name	Description
"Content:"	CONTENT	The actual contents of the e-mail.
"Content-Type:"	CONTENTTYPE	MIME-content type of the content (e.g. text/html)
"MIME-Version:"	MIME-VERSION	This means the post is a MIME post and the line tells you the version, so your client knows what MIME version it is.
"Date:"	DATE	Contains the date when the post was sent to a server for the first time.
"Received:"	RECEIVED	Verified name of the sending computer and in brackets the name and the IP Address contributed by the message envelope.
"X-Mailer"	X-MAILER	Contains the client that was used for creating the post
"X-Accept-Language:"	X-ACCECPT-LANGUAGE	Contains the language code of the language in that the poster would like to be contacted when replying (e.g. "en" means English), can also contain more than one language code.
"Subject:"	SUBJECT	Contains the subject of the post
"Message-ID:"	MESSAGE-ID	Contains a worldwide unique ID to identify the post. Server use it when exchanging messages. It usually has the following format: <unique-string-or-number@server.com>

RFC 822 Headers containing email-addresses:

POP3 Header	Internal Name	Description
From:	FROM	The line with name/e-mail of the poster who wrote the message.
To:	TO	The line with name/e-mail of the receiver of the message.
Reply-To:	REPLY-TO	This line contains name/e-mail address that should be used for replying to the post by mail. If no such line exists, the data of the From-line is used.
Return-Path:	RETURN-PATH	Sender address contributed by the message envelope.
Cc:	CC	Carbon copy (same as To:)
Bcc:	BCC	Blind carbon copy (list of recipients will not appear)
Apparently-From:	APPARENTLY-FROM	According to the sendmail book, the "Apparently-From:" header is added by Smail 3.x when it cannot find any of the valid sender headers in the message. The field used on "Apparently-From:" is taken from the the message envelope. Sendmail will create a "From:" header in this situation.
Apparently-To:	APPARENTLY-TO	Same as Apparently-From but refers to the recipient.
Sender:	SENDER	Contains name/e-mail of the person who sent this post. That needn't be the author of the message (you may post a message for another person). In case poster and sender is the same person, there is only a From-line

15.5 Data templates for the E-Mail template

The Data templates below the E-Mail template (derived from class "EMAIL") are derived from the respective Data classes with the following overloaded attributes:

MIME type (MIMEType)

This attribute contains the match criterion **POP3($MATCH_LOCAL$,MIME)** in its "Type Property" field so the Adapter can instantiate this template if the MIME type in the e-mail is equal to the contents of this attribute (e.g. text/plain).

<DataTypeName> **(TargetName)**

The other attribute (with the internal name TargetName) contains the expression **POP3(CONTENT)** to fill the attribute with the contents of the e-mail and an expression of type **MIME(***<MimeType>***)** to pick the part with the specified MIME typ in the multipart message (e.g. MIME(image/gif) to extract the GIF image). The name of this attribute depends on the document type (e.g. ASCII, AFPDS, HTML, ...).

15.6 Setup of the POP3 Receiver

To build a new POP3 Receiver setup, duplicate the template "Adapter (POP3)", which can be found in the Library folder "Library / Adapters".

Add a reference of a Role to the new Adapter to authorize it to perform its tasks.

Add a Queue to the Adapter, which will be used as inbox for incoming e-mails.

Select the Mailbox Configuration object "POP3 Mailbox" below the POP3 Receiver and use its attributes to define the e-mail account that should be used by the POP3 Receiver.

Note that the Adapter can only map those media types into child objects below the E-Mail instances which are already defined below the E-Mail template.

Instantiate the new Adapter template on the Node where the POP3 Receiver should run.

To start the new Adapter select it and click the button "Start adapter", which automatically starts the POP3 Receiver below.

15.7 Components of the POP3 Receiver

15.7.1 POP3 MAILBOX

The POP3 Receiver requires at least one Mailbox Configuration object, which contains all attributes required to define an e-mail account. The attribute values are entered directly in the instance.

15.7.1.1 States

Ready	In this state POP3 Mailbox waits for incoming messages.
Fetching	POP3 Mailbox is fetching e-mails (processing state).
Error	If the last query failed POP3 Mailbox changes its state to "Error" (if the POP3 Receiver would change to "Error" the whole setup would be halted). As errors should not stop the process of receiving e-mails perpetually (e.g. error caused by temporary network problem) POP3 Mailbox automatically changes its state from "Error" to "Fetching" after the interval defined via the attribute "Poll interval" and stores the error message in the attribute "Last Error".

15.7.1.2 Attributes

Hostname	Name or IP address of the POP3 server.
User	User name of the mail account.
Password	Password of the mail account.
Poll interval [s]	Defines the call interval in seconds after which e-mails are automatically fetched from the POP3 server. If the default value 0 is specified, the e-mails have to be fetched manually by selecting the POP3 Receiver and clicking the button "Check now".
Leave messages on server	If checked, the downloaded messages are not deleted on the POP3 server.
Get attachments	If checked, the attachments are downloaded and forwarded to the Adapter.
Get raw MIME data	If checked, the POP3 Receiver sends the entire MIME stream (saved as a temporary file) to the Adapter (only required for forwarding and recognition). As the attribute "Raw MIME data" of the E-Mail object derived from class "EMAIL" has the expression **POP3(RAWMIMEDATA)** defined in its field "Type property", it will be used to save the e-mail.
UIDL	This is a unique identifier from the POP3 server that allows the Receiver to keep track of the messages already fetched. This is an optional POP3 feature and is only used if the messages should not be deleted on the POP3 server.
Last Error	Stores a description of the last error, this attribute is only visible if the POP3 Mailbox is currently in the state "Error".

Note: Via the method "Check Now" of the POP3 Receiver it is possible to fetch e-mails immediately without waiting for the next interval. This method is available after the POP3 Receiver has been automatically started by the Adapter as the Pre constraint is set to "EngineStatus==1".

15.7.1.3 Methods

There are no methods defined.

15.7.2 E-mail object

A template derived from the class "EMAIL", which is a child class of "PHYSICAL DOCUMENT", is located in the General Container of the POP3 Receiver to enable it to generate one E-mail object for each incoming e-mail. The attributes are filled with general information of the e-mail header lines (first section of an e-mail containing information about sender, recipient, etc.). Via expressions in the field "Type property" containing the name of the header line in the e-mail these attributes are filled by the Adapter, e.g. the attribute "Content type" contains the match criterion POP3(Content-Type) to fill the attribute with the value of the header line "Content-Type:".

15.7.2.1 States

Ready (1)	This state has been defined to avoid that the state machine of the base class is taken.

15.7.2.2 Attributes

Content Type	Stores the content type of the e-mail (e.g. multipart/mixed) via the following expression in the field "Type Property": POP3(Content-Type)
Subject	Stores the subject of the e-mail via the following expression in the field "Type Property": POP3(Subject)
From	Stores the sender of the e-mail via the following expression in the field "Type Property": POP3(From)
To	Stores the receiver of the e-mail via the following expression in the field "Type Property": POP3(To)
CC	Stores the e-mail addresses put on copy via the following expression in the field "Type Property": POP3(Cc)
BCC	Stores the e-mail addresses put on blind copy via the following expression in the field "Type Property": POP3(BCC)
Date	Stores the send date of the e-mail via the following expression in the field "Type Property": POP3(Date)
Raw MIME data	As each attachment and media type is stored in a separate Data object by the Adapter it would not be possible to forward the e-mail to other mail clients or to perform a recognition, therefore the attribute "Raw MIME data" stores the whole e-mail via the following expression in the field "Type Property" if option "Get raw MIME data" has been selected in the Mailbox Configuration object: POP3(RAWMIMEDATA)
Account	Currently not in use.
Return path	Stores the sender address contributed by the message envelope via the following expression in the field "Type Property": POP3(Return-Path)

Content Type	Stores the content type of the e-mail (e.g. multipart/mixed) via the following expression in the field "Type Property": POP3(Content-Type)
Received	Stores the verified name of the sending computer and in brackets the name and the IP address contributed by the message envelope via the following expression in the field "Type Property": POP3(Received)
Message ID	Stores the worldwide unique ID to identify the e-mail via the following expression in the field "Type Property": POP3(Message-ID)
MIME version	Stores the MIME version used in the e-mail via the following expression in the field "Type Property": POP3(MIME-Version)
references	Currently not in use.
Match Me	The match criterion **POP3($MATCH_ITERATION$,matchme)** in the field "Type Property" of attribute "Match Me" is used by the Adapter to decide if the E-Mail template may be instantiated. Every time the POP3 Receiver gets an e-mail it forwards the additional name/value pair "matchme=message" to the Adapter, to signalize that there is an incoming e-mail. As the value of attribute "Match Me" is identical with the value the Adapter received, the e-mail template will be instantiated. Keyword **$MATCH_ITERATION$** works the same as $MATCH_LOCAL$, i.e. only the e-mail template without its child objects will be instantiated, but in addition it enables the Adapter to instantiate each child of the e-mail template repeatedly. The child objects of the e-mail template have match criteria with keyword $MATCH_LOCAL$ defined, but due to the parent match criterion with keyword $MATCH_ITERATION$, they not only get instantiated once but as often as required. If for instance an e-mail contains three GIF images, the Adapter would instantiate the matching (via keyword $MATCH_LOCAL$) GIF Data object three times.

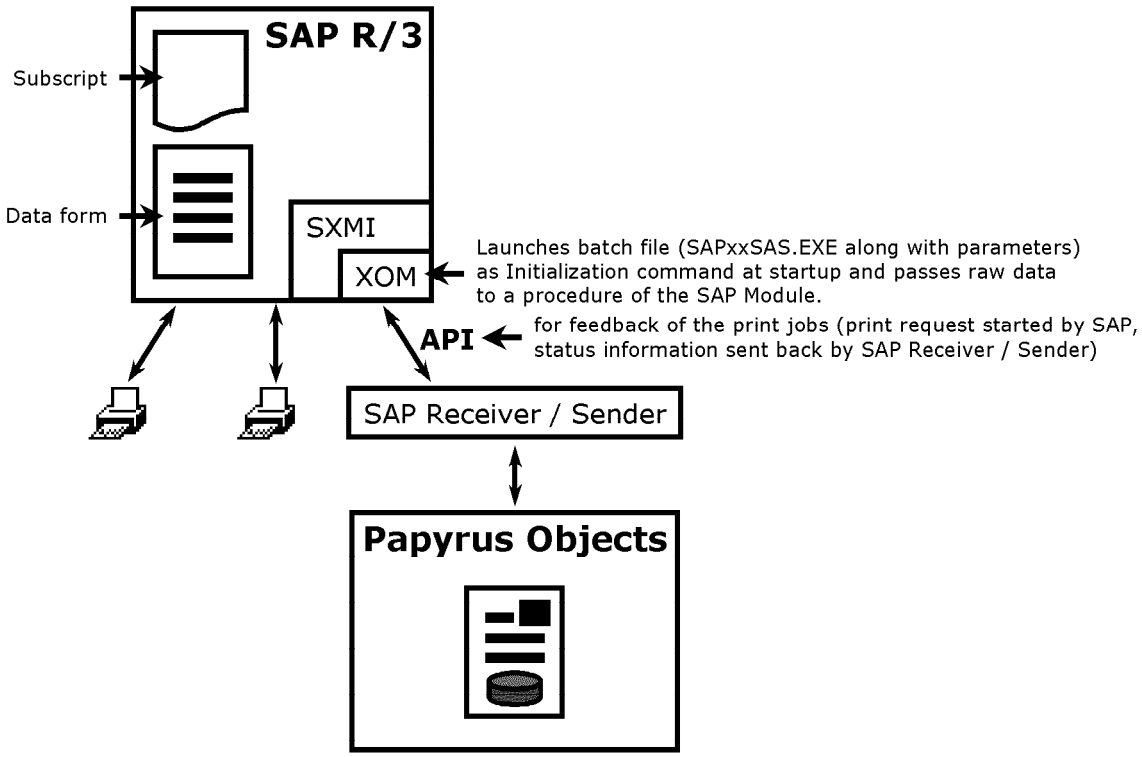
Note: Some attributes have additional match criteria starting with the string XML, thus they can be used by XML Adapter setups or a POP3 Adapter setup using PQL.

15.7.2.3 Methods

View email	Desktop View method to display the e-mail.

16.0 SAP Receiver / Sender

The objective is to receive output from SAP and control these SAP jobs (via states). Therefore Papyrus Objects and the SAP Receiver / Sender act as an external Output Management System to SAP. The **SAP Factory Adapter** has the responsibility of **creating / starting SAP Adapters** to communicate with all running on SAP R/3 instances. Thus the SAP Factory Adapter needs **one SAP Adapter template for each SAP R/3 instance**. The SAP Factory Adapter is then able to instantiate a matching Adapter template, which is used to communicate with the respective SAP instances.


SAP R/3
Subscript
Data form
SXMI
XOM
Launches batch file (SAPxxSAS.EXE along with parameters)
as Initialization command at startup and passes raw data
to a procedure of the SAP Module.
API
for feedback of the print jobs (print request started by SAP,
status information sent back by SAP Receiver / Sender)
SAP Receiver / Sender
Papyrus Objects

Benefits of the SAP Receiver / Sender at a glance

- Consistent interfaces using SAP's standard interface SXMI and XOM.
- Integration with business critical applications using SAP's RDI.
- Centralized management and control. When issues occur the administrator only has to work with one single focal point - Papyrus Objects.
- It is the perfect tool for planning distributed output strategy. The SAP Receiver / Sender can easily be incorporated into the Papyrus Objects workflow.

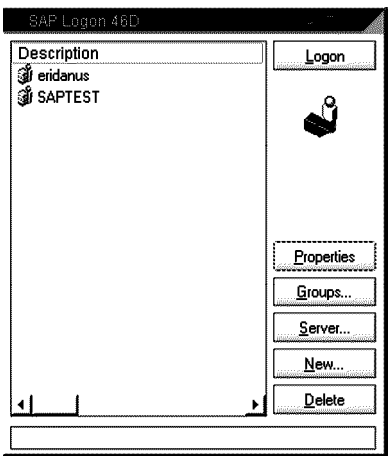
The following setup tasks have to be performed

- Installation of ISIS components in SAP R/3
- Configuration of components in SAP R/3
  - Real Output Management System (ROMS)
  - Logical Output Management Systems (LOMS)
  - Output Devices
- Customization of Ready-to-use SAP Adapter setup in Papyrus Objects

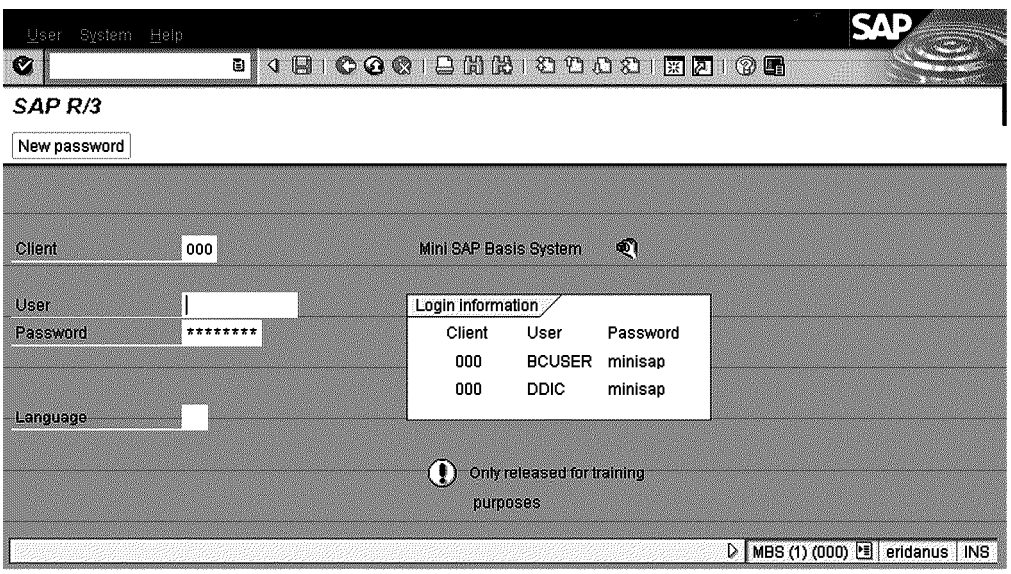
Note: The SAP Receiver / Sender is currently available for Windows 32 bit operating systems. For other platforms contact the ISIS Marketing team.

16.1 Settings on SAP R/3

Log on to SAP R/3.

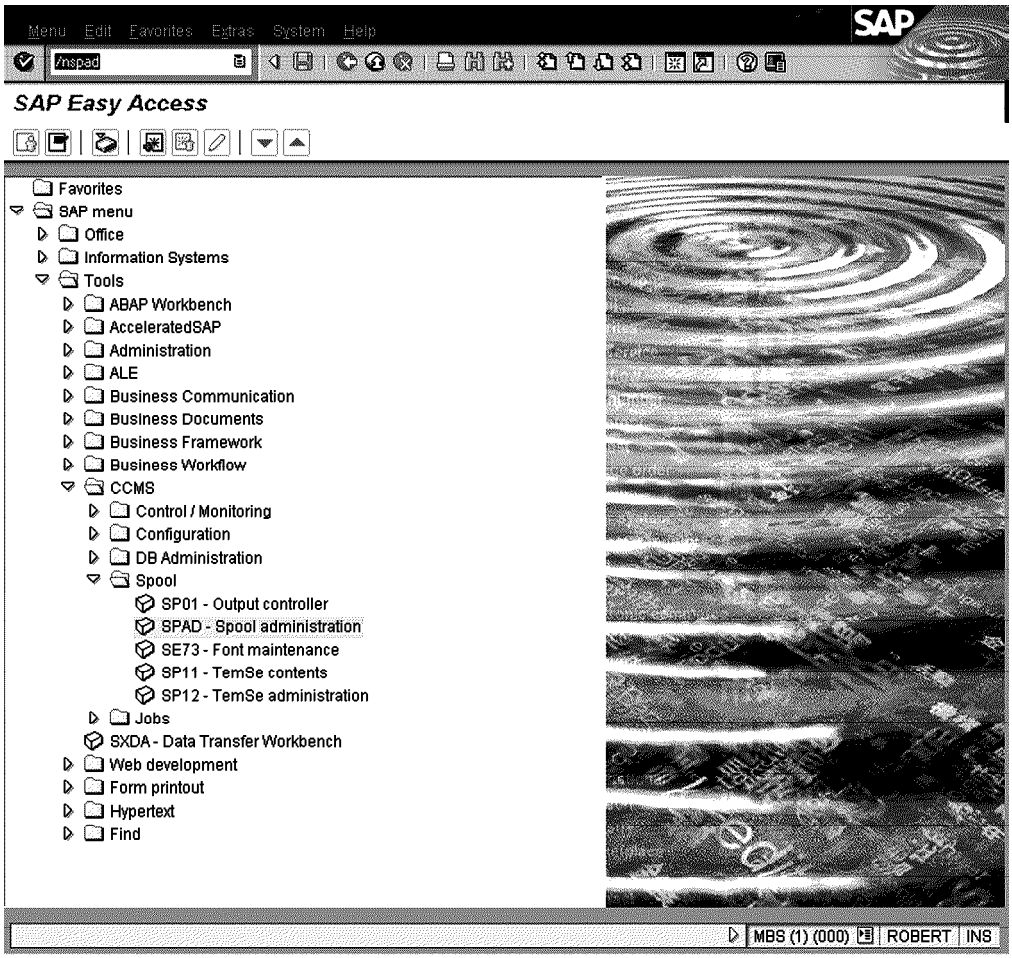


Log on using the user name and password provided by the System Administrator. Default language is english (EN), alternative languages are available for example DE for German.



16.1.1 SAP Easy Access

Change to **SAP Easy Access** and select **/nspad**, which is a SAP transaction that contains the definition of the *External Output Management System* (in this case Papyrus Objects) to link to it, from the listbox or select it in the tree below.



Real Output Management System

The definition of the Real Output Management Systems (ROMS) contains a reference to all the functionality offered by Papyrus Objects and the SAP Adapter. The definition of the Real Output Management System contains all parameters for connection required by all Logical Output Management Systems (LOMS).

Logical Output Management System

To be able to define an Output Management System for each department Logical Output Management Systems (LOMS) are required. Logical Output Management Systems are created specifically to communicate directly with the SAP Adapter. Logical Output Management Systems may use all the functionality of the Real Output Management System or only some of the functionality (e.g. one department could have 3 PCL printers and not the right to query another one 5 IPDS printers with the right for queries).

16.1.2 Spool Administration : Real Output Management System

All parameters of the Real Output Management System are set up (e.g. in Job status it is defined whether queries can be made, jobs may be deleted, etc.), the respective Logical Output Management System may only use a subset of what is defined here. The **Initialization command** in the section "SAP configuration" contains the path and the file name to a batch file (*.cmd) which is sent to the SAP Factory Adapter and starts the SAP Sender (SAS). This batch file should be placed into the working directory of SAP where the SAP instance is located (e.g. if the System ID is MBS the directory name will be the same after the installation). The contents of the first option (/CMD) is used by the SAP Adapter that handles the respective service of SAP as match criterion to instantiate the appropriate Task template. The contents of all other options is filled into Task attributes by the SAP Adapter.

Enter the Host name, the System ID, and the System number combined by underscores in the field **Initialization instance** (e.g. ROBERT_MBS_00 if ROBERT is the Host name, MBS the System ID, and 00 the System number). The contents of the field Initialization instance is stored in the variable &ES.

Example for a batch file for the Initialization command:

```
pocw3sas.exe /CMD=INITINST /HOST=PCName /PORT=19194 /FACTORYPORT=19192
             /USER=DDIC /PWD=minisap /CLIENTID=000 /SAPHOST=PCName
             /ESS=MBS /SYSNBR=00 /LANG=EN /ROMS=objectS /DBG=Y
```

The assigned value of the following option is used by the SAP Factory Adapter as a match criterion for the instantiation of Adapters:

/CMD=INITINST

The following options are available:

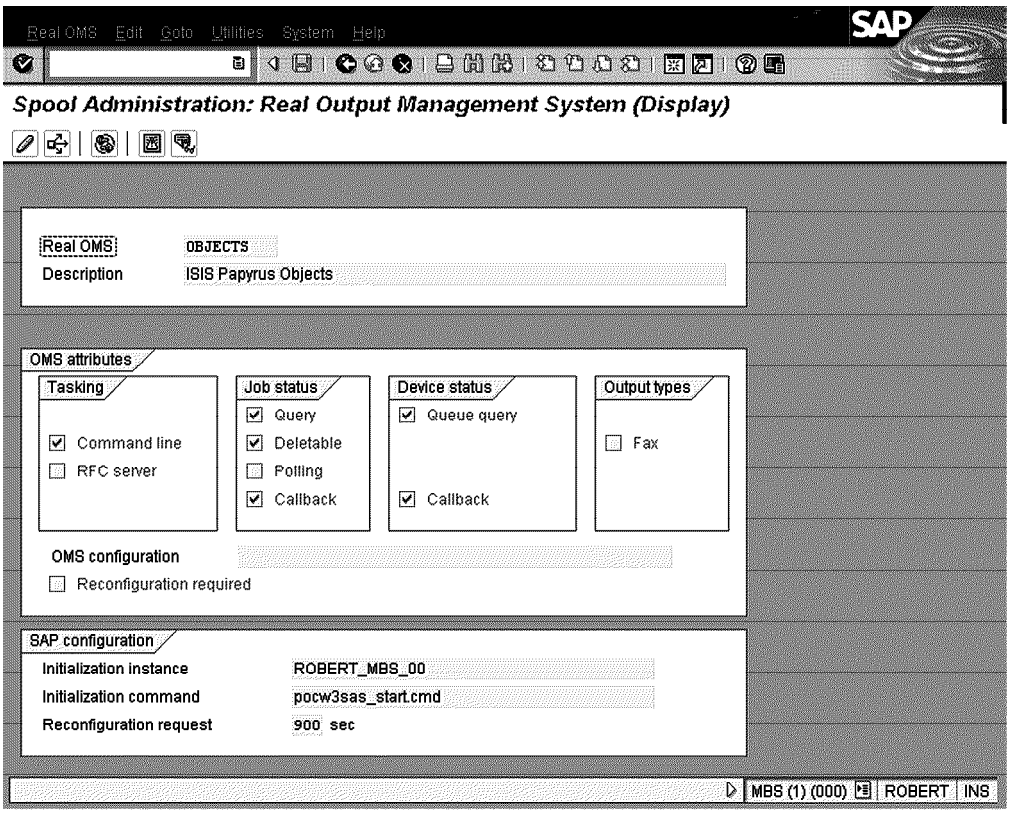
/HOST	Name of the PC Papyrus Objects is running, used by the instantiated Adapter and by the Factory Adapter as well to communicate with SAP
/PORT	Port the instantiated Adapter will use for communication with SAP.
/FACTORYPORT	Port the Factory Adapter will use for communication with SAP.
/USER	User name required by the instantiated Adapter and the Factory Adapter to log on SAP.
/PWD	Password required by the instantiated Adapter and the Factory Adapter to log on SAP.
/CLIENTID	Sequential number for the Client connecting to SAP R/3 (Papyrus Objects)
/SAPHOST	Name of the PC where SAP R/3 is running
/ESS	System ID (ID of calling the SAP R/3 system)
/SYSNBR	Sequential number for the SAP instance
/LANG	Language for dialog information (currently not in use)
/ROMS	Name of the Real Output Management System (case sensitive), when Receiver sends back status messages it has to know which ROMS to address.
/DBG	Debug flag, if set to Y the log file "pocxxsas.txt" is generated in the subdirectory SAPSEND which is located in the working directory of the Papyrus Objects Kernel

Structure of the log file "pocxxsas.txt"

- The first lines show which parameters have been passed to the program, all variables (e.g. &EL) are resolved to be able to check the SAP setup.
- TCP/IP messages show if sending was successful
- POST section shows the whole spool file from option /F encoded in Base 64 (to avoid special characters)
- TCP/IP messages show if data exchange was successful
- If SAP Receiver sent something back the header "POCXXXSAP" appears along with the information sent back (e.g. Task ID, Job ID, ...)
- TCP/IP status information

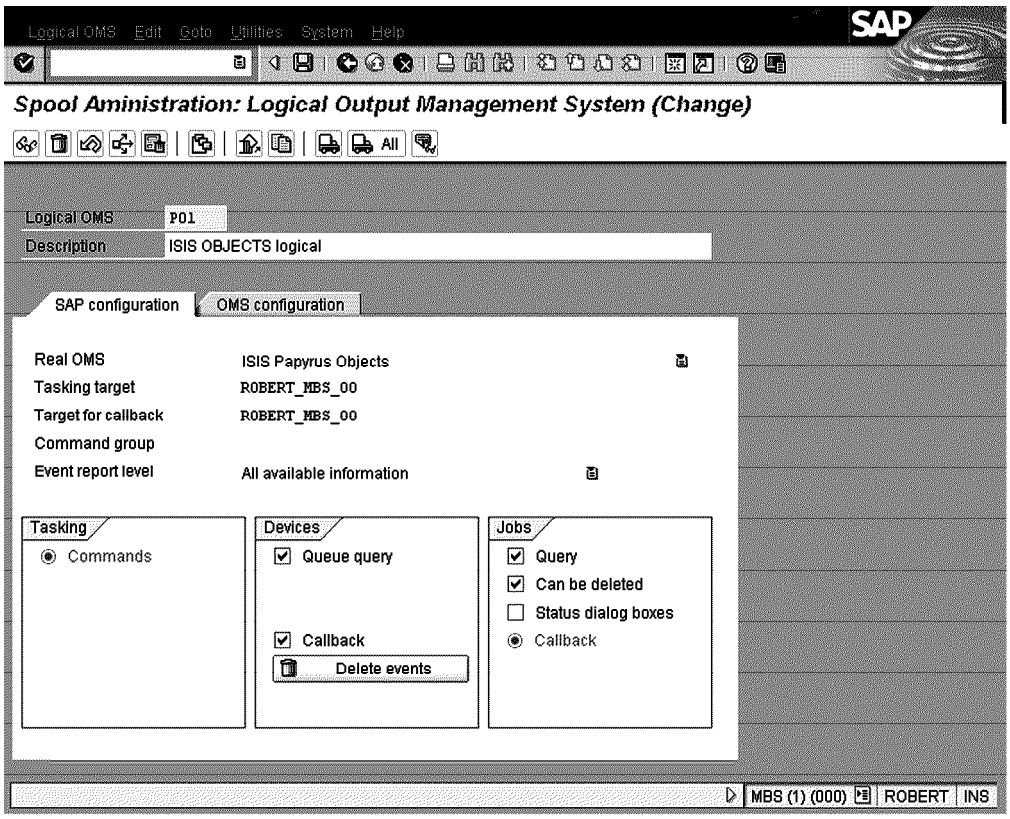
Reconfiguration request is the polling interval for communication (e.g. send callback every 100 seconds).

The checkbox **Reconfiguration required** specifies whether changes of the configuration should be updated within the SAP Adapter. The SAP Adapter will know to reconfigure itself after the next interval time if a Reconfiguration request is made.



16.1.3 Spool Administration : Logical Output Management System - SAP Configuration

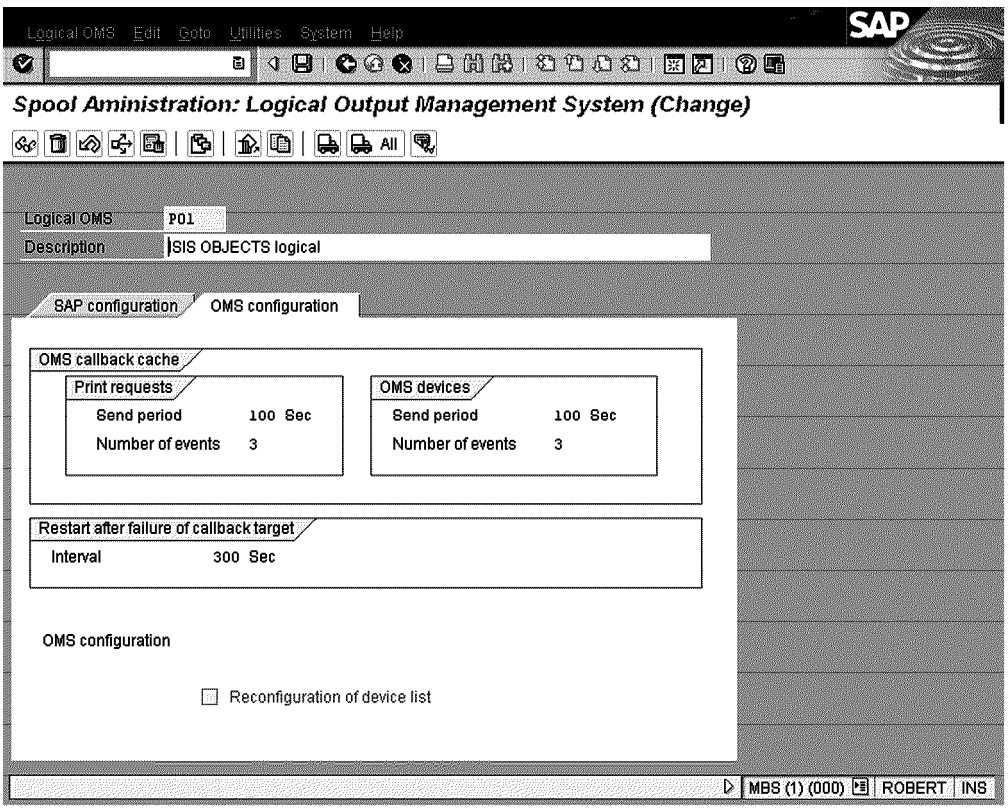
Click on the tab **"SAP Configuration"** to enter the name of the Real Output Management System (in our case Papyrus Objects) in the field **Real OMS**. This name is used as a link to the Output Devices of this Logical Output Management System.



16.1.4 Spool Administration : Logical Output Management System - OMS Configuration

Click on the tab **"OMS Configuration"** to specify the following parameters, which are intended to keep network traffic low.

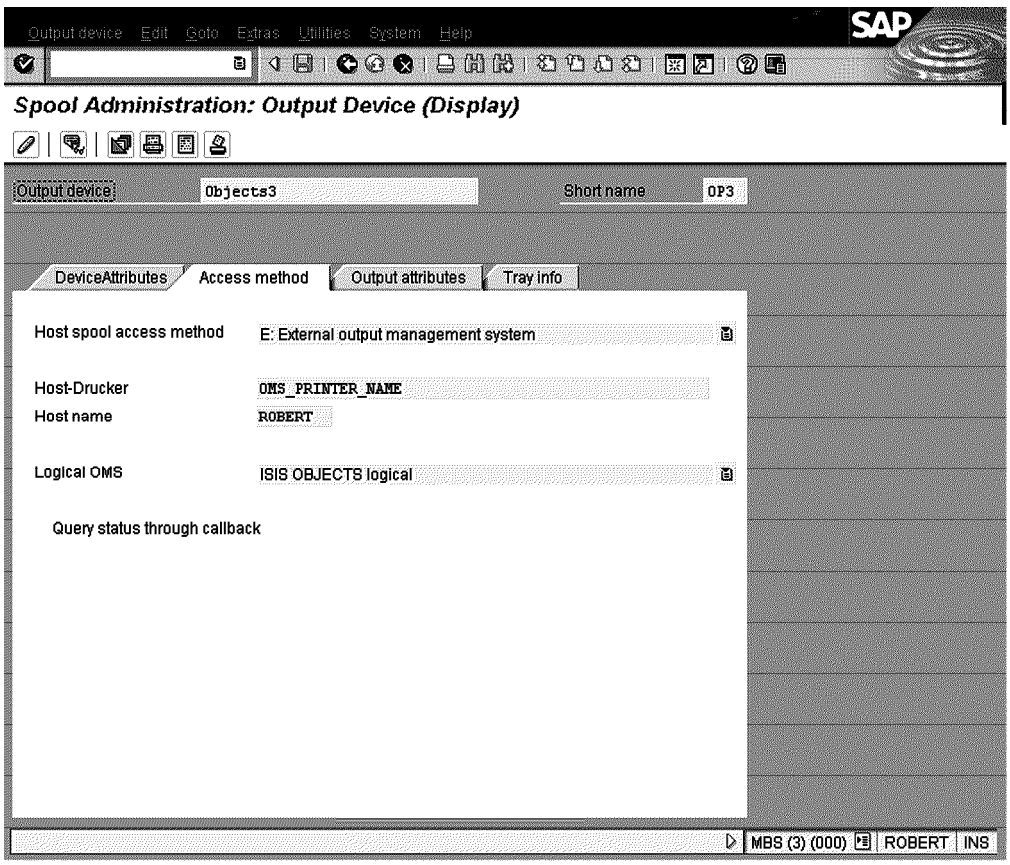
Send period	defines how long the Adapter should wait until it sends a callback to the SAP instance.
Number of events	defines how many events the Adapter has to wait for until it sends a callback to the SAP service.



16.1.5 Spool Administration : Output Device

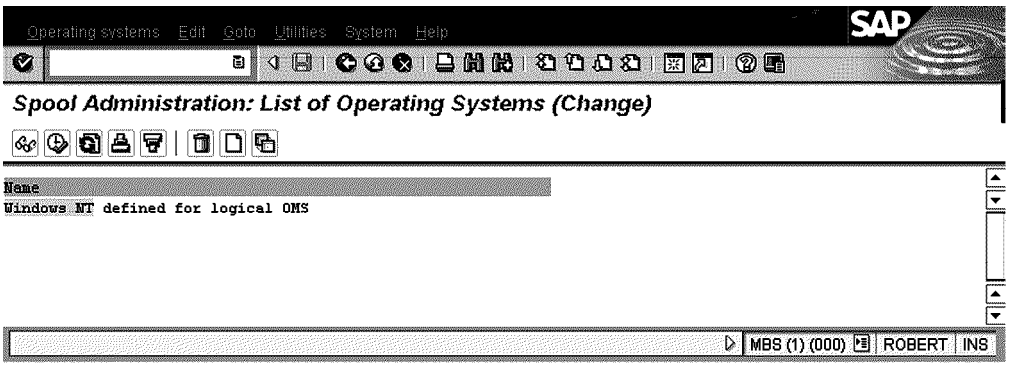
Instead of the Spool Administrator creating Output devices to print specifically to a printer, output devices will be created and associated to Logical Output Management Systems, which will be associated to an External Output Management System (in our case Papyrus Objects).

Select the option E for "External output management system" from the listbox **Host spool access method**, only after this selection the field **"Logical OMS"** appears. Enter the name of the Logical Output Management System to link to the Logical Output Management System the devices is associated with (see above). In the field **Host printer** enter the name of the device, this name will mapped into the SUBMIT command below via the variable &P.



16.1.6 Spool Administration : List of Operating Systems

Now command lines for the **SAP Sender Module** (pocwsas.exe) have to be entered for the **SUBMIT command**. To get to the command section for Windows NT double click the entry **Windows NT**.



16.1.7 Spool Administration : Operating Systems Commands

Here command lines are defined for actions. The batch file can be also used for Logical Output Management Systems to send spool jobs. Only the Submit command contains a parameter for the match criterion for instantiation.

Submit

```
pocw3sas.exe /CMD=SUBMIT /HOST=PCName /PORT=19194 /EI=&EI /EG=&EG /P=&P
             /F=&F /FORM=RDI /ESS=&Es /ES=&ES /L=&L /C=&C /DBG=Y
```

Queue query

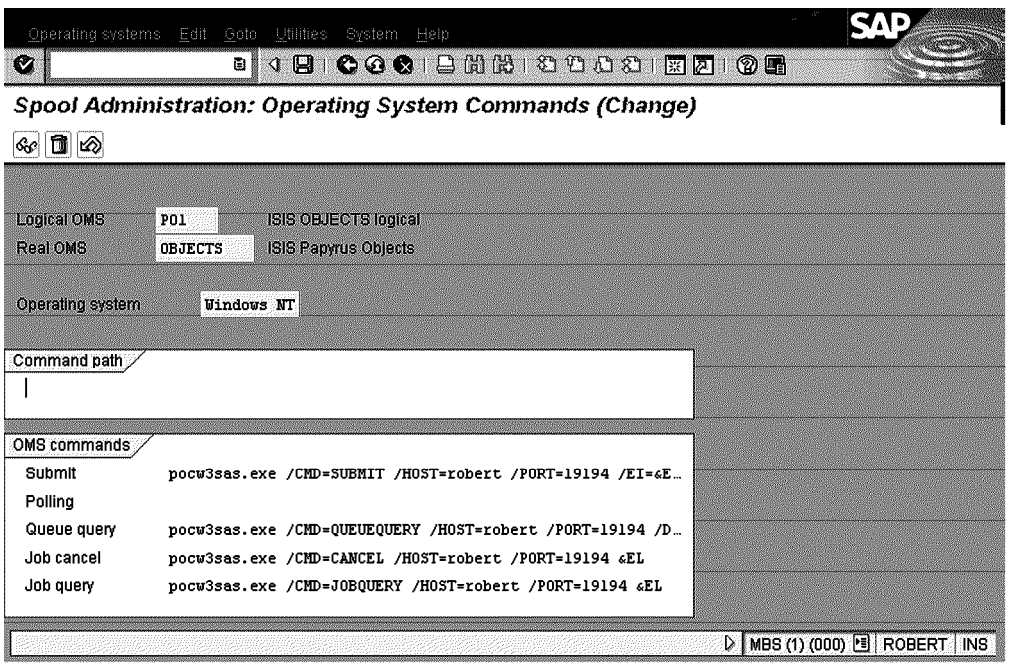
```
pocw3sas.exe /CMD=QUEUEQUERY /HOST=PCName /PORT=19194 /P=&P /ESS=&Es
```

Job cancel

```
pocw3sas.exe /CMD=CANCEL /HOST=PCName /PORT=19194 &EL
```

Job query

```
pocw3sas.exe /CMD=JOBQUERY /HOST=PCName /PORT=19194 &EL
```


Operating systems Edit Goto Utilities System Help
SAP
Spool Administration: Operating System Commands (Change)
Logical OMS P01 ISIS OBJECTS logical
Real OMS OBJECTS ISIS Papyrus Objects
Operating system Windows NT
Command path
OMS commands
Submit pocw3sas.exe /CMD=SUBMIT /HOST=robert /PORT=19194 /EI=&E…
Polling
Queue query pocw3sas.exe /CMD=QUEUEQUERY /HOST=robert /PORT=19194 /D…
Job cancel pocw3sas.exe /CMD=CANCEL /HOST=robert /PORT=19194 &EL
Job query pocw3sas.exe /CMD=JOBQUERY /HOST=robert /PORT=19194 &EL
MBS (1) (000) ROBERT INS

The assigned value of the following option is used by the SAP Adapter as a match criterion for the instantiation of Task templates:

/CMD=Submit

The following options are filled into Task attributes by the SAP Adapter:

/EI	Spool ID
/EG	RMG (Reply message group)
/P	Printer
/F	Spool file (the actual data)
/FORM	RDI or ASCII
/ES	SAP instance
/L	Format type
/C	Number of copies
/DBG	Debug mode
/ESS	System ID

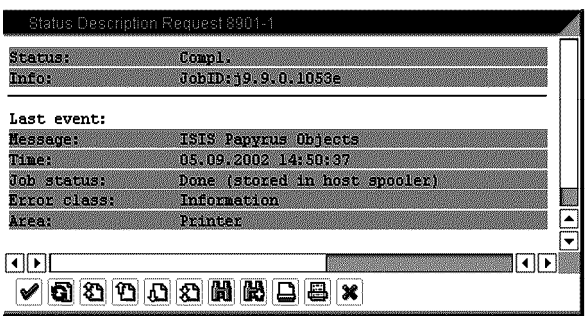
The following variable is used directly by the SAP Module:

&EL	Variable &EL contains a list of jobs, which is sent to the listener port of the SAP Module directly

The field **Command Path** can be used to change the path of the SAP Receiver Module (pocw3sap.exe).

16.1.8 Transaction SP01

Section Transaction SP01 is used for spool information



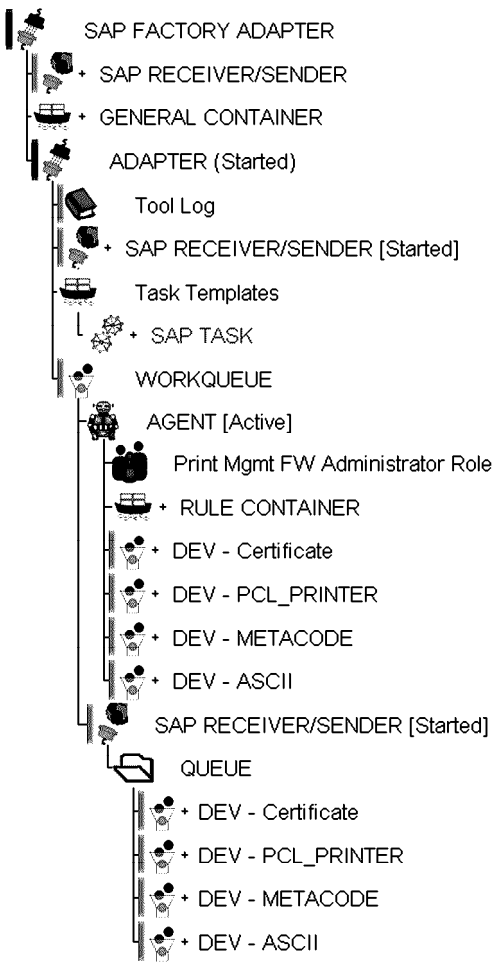
16.2 Setup on Papyrus Objects

The SAP Factory Adapter has to be started first. It will subsequently start the SAP Receiver / Sender automatically. The SAP Factory Adapter will then instantiate appropriate SAP Adapter templates, one for each SAP R3 instance. Each instantiated Adapter is used to handle the communication with the respective SAP instance.

Via the **Initialization command** SAP R/3 starts a batch file that sends a message to the Papyrus SAP Factory Adapter with various options. The contents of the first option (/F) is used by the SAP Adapter that handles the respective instance of SAP R/3 as match criterion to instantiate the appropriate Task template.

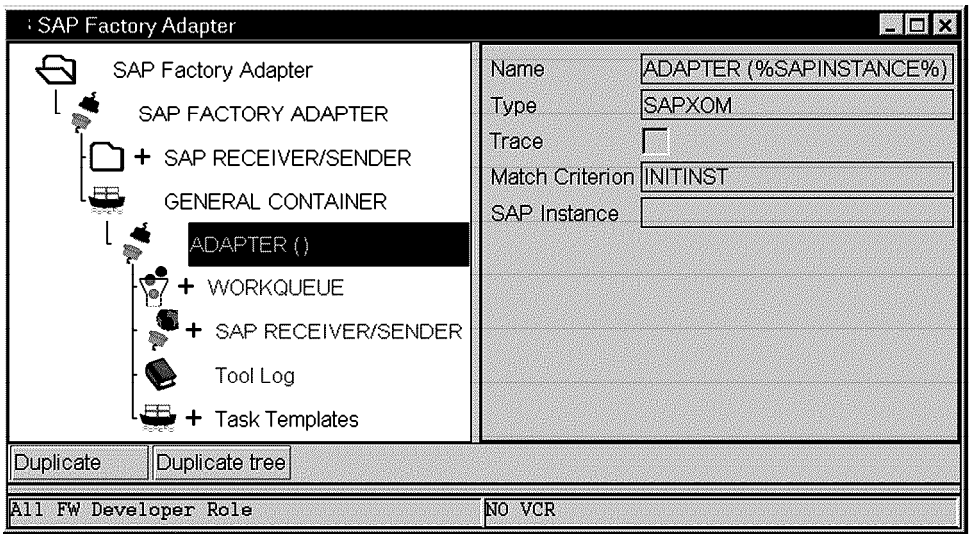
Possible values of **option /FORM** are /FORM=RDI or /FORM=ASCII (default: F=RDI). If **"RDI"** is passed the SAP Sender Module (pocw3sas.exe) uses the characters from position 21 - 38, of the first line in the spool file (Layout name) as value for the form. The SAP Receiver Module (pocw3sap.exe) then sends **"FORM=NAMEOFFORM"** to Papyrus Objects. If **"ASCII"** is passed the SAP Receiver Module sends **"FORM=ASCII"** to Papyrus Objects. Note that RDI data can also be processed by Papyrus DocEXEC.

The contents of all other options is filled into attributes by the Adapter.



Instantiation of SAP Adapter templates by the SAP Factory Adapter

The SAP Factory Adapter instantiates a matching SAP Adapter template. The SAP Adapter template is matching if the attribute "Match Criterion" of the SAP Adapter template which contains the "Type Property **SAPXOM($MATCH_RECURSIVE$,CMD)**", has the same value as the value of **option CMD** sent by the **Initialization command**. The attribute "SAP instance" is filled by the SAP Factory Adapter with the name of the SAP instance (/ESS option of the Initialization command), this way the SAP Adapter / Receiver is identified for the respective SAP instance.



Processing of Tasks

The SAP Adapter instantiates a matching Task template into its Work Queue. The Task template is matching if the attribute "Match Criterion" of the Task template, which contains the "Type Property **SAPXOM($MATCH_RECURSIVE$,CMD)**", has the same value as the value of **option CMD** sent by the **SUBMIT command**. In the Work Queue there is another reference to the SAP Receiver / Sender which contains the Tool method "state constructed". Since the Task contains a Material method with the same "Internal method name" the Agent will start this tool/material method. The tool/material method "state constructed" sends the Connection ID to SAP. The objective is to let SAP know which Task processes the SAP job. After this the SAP Module sends the Program Event "Goto_Waiting" which changes the state of the Task from "Empty" to "Waiting". As the state of the Task is "Waiting" the Move Tool can move the Task into the appropriate Processing Queue (in this sample there is only one Processing Queue). The Task will then be moved into the appropriate Destination Queue by the Agent if the contents of the attribute "Destination queue" of the Task matches the attribute "Destination queue" of the Queue this is because the Queue contains the match criterion **AGENT(MUST_MATCH)** in its "Type Property" field. All further steps are related to generating output and/or printing.

16.2.1 Attributes of the SAP Factory Adapter and the SAP Adapter templates

Type

The attribute "Type" is used in match criterions to specify which Adapter should be used for instantiation or filling attributes.

Match Criterion

Contains the match criterion SAPXOM($MATCH_RECURSIVE$,CMD) in the "Type Property" field used for instantiation of SAP Adapter templates or SAP Task templates.

16.2.2 Attributes of the SAP Receiver / Sender

Listener Port
This port allows SAP R/3 to communicate with Papyrus SAP Receiver / Sender. The SAP Factory Adapter fills this attribute with the value of **option PORT** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(PORT)** in the "Type Property" field.

Client

Client ID used to log on to SAP R/3. The SAP Factory Adapter fills this attribute with the value of **option CLIENTID** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(CLIENTID)** in the "Type Property" field.

User

User name to be used to allow the SAP Receiver / Sender to log on SAP R/3. The SAP Factory Adapter fills this attribute with the value of **option USER** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(USER)** in the "Type Property" field.

Password

Password required to log on SAP R/3. The SAP Factory Adapter fills this attribute with the value of **option PWD** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(PWD)** in the "Type Property" field.

Language

SAP R/3 can be run in various languages. This attribute determines which language should be used for this session. The SAP Factory Adapter fills this attribute with the value of **option LANG** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(LANG)** in the "Type Property" field.

Hostname

The name of the server on which SAP R/3 is installed. The SAP Factory Adapter fills this attribute with the value of **option SAPHOST** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(SAPHOST)** in the "Type Property" field.

System number

The System number identifies the host computer system. The SAP Factory Adapter fills this attribute with the value of **option SYSNBR** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(SYSNBR)** in the "Type Property" field.

Gateway host and **Gateway service**

These parameters ensure that SAP Receiver / Sender obtains a connection to SAP R/3 using the DIAG interface. Used if SAP R/3 runs behind a gateway program; otherwise no connection to SAP R/3 will be established.

Real OMS Name

Name of the **R**eal **O**utput **M**anagement **S**ystem (case sensitive!). The SAP Factory Adapter fills this attribute with the value of **option ROMS** of the **Initialization command** as the attribute contains the match criterion **SAPXOM(ROMS)** in the "Type Property" field.

Is Factory

If this flag is set the SAP Receiver / Sender is used for the SAP Factory Adapter; otherwise it is used for the SAP Adapter.

The following attributes contain the state numbers of the respective states in Papyrus Objects to be able to send the proper state name back to the SAP R/3 User. state numbers have to be two digits long (leading zero if smaller than 10). If more than one state number in Papyrus Objects is related to the same state name in SAP, state numbers have to be separated by blanks.

state Completed

02

state Error

03

state Waiting

00

state Running

01 06

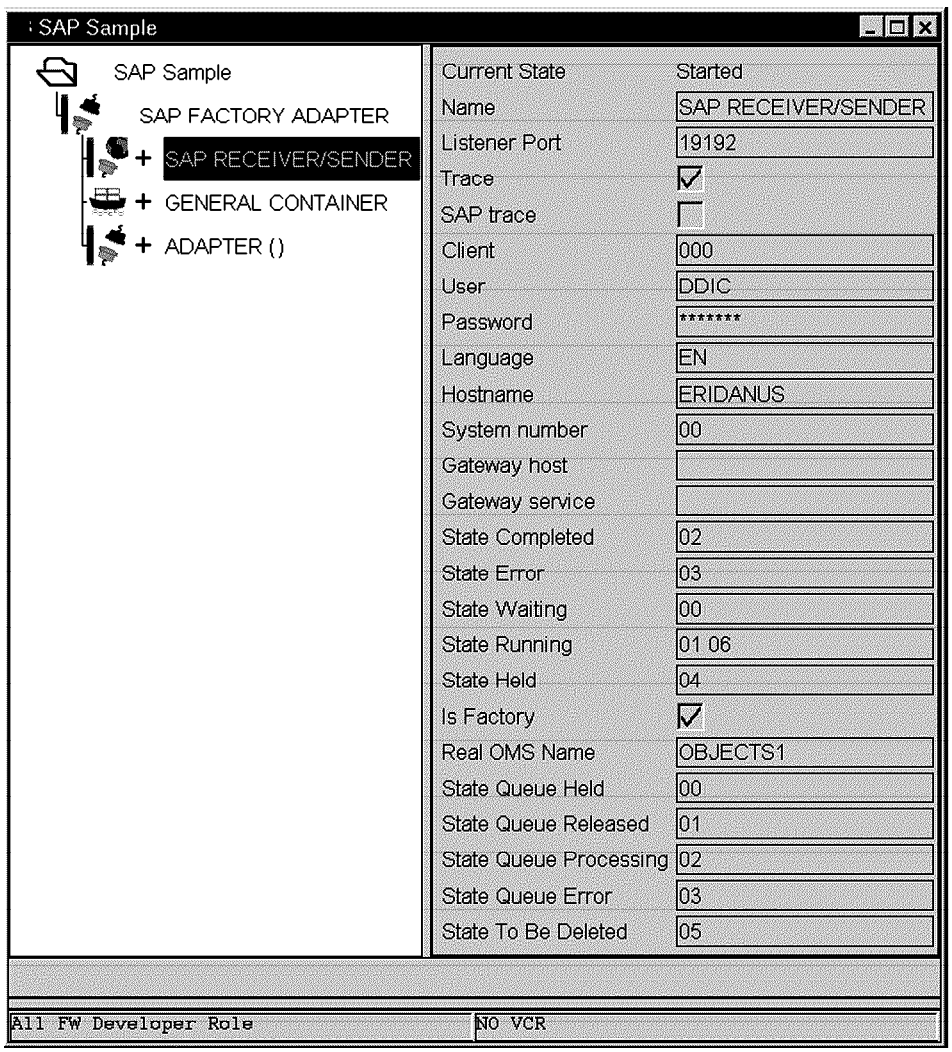
state Held

04

...

Move target (MoveTarget)

The tool/material method "state constructed" can only be started by the Agent if the attribute "Move target" of the SAP Task template and the SAP Receiver / Sender template contain the same string (in the Ready-To-Use Setup the string "Start" is defined). Otherwise the tool/material method "state constructed" would be locked via the match criterion **AGENT(MUST_MATCH)** defined in the field "Type property" of the attribute "Move target" of the SAP Task template.



16.2.3 Attributes of the SAP Task

For an easier identification of each SAP Task instance generated by the SAP Receiver, the Visible name of the SAP Task template can be changed in the Template Editor. By entering the Internal name of the attribute "SAP Title" between percentage signs (e.g. Visible name = Task - %SAPTitle%) in the field "Visible name", each instantiated SAP Task will display the output request comming from SAP. For details about general attributes derived from the Base class "TASK" see chapter *"Task"*:

Distribution queue (DistributionQueue)

This attribute can be used in Rules to let the Agent move incoming SAP Tasks into the Distribution Queue. Enter the same string that is defined in the attribute "Move target (MoveTarget)" of the Queue the Task should be moved into.

Completed queue (CompletedQueue)

This attribute can be used in Rules to let the Agent move finished SAP Tasks into the Completed Queue. Enter the same string that is defined in the attribute "Move target (MoveTarget)" of the Queue the Task should be moved into.

Error queue (ErrorQueue)

This attribute can be used in Rules to let the Agent move erroneous SAP Tasks into the Error Queue. Enter the same string that is defined in the attribute "Move target (MoveTarget)" of the Queue the Task should be moved into.

Processing Queue (ProcessingQueue)

This attribute can be used in Rules to let the Agent move SAP Tasks into certain Processing Queues. Enter the same string that is defined in the attribute "Move target (MoveTarget)" of the Queue the Task should be moved into.

Move target (MoveTarget)

In Rules this attribute is assigned with the contents of one of the attributes "Processing queue", "Distribution queue", "Completed queue" or "Error queue" depending on the current state of the SAP Task or any other condition. Because of the match criterion **AGENT(MUST_MATCH)** in the field "Type property" of this attribute, the Agent will move the SAP Task into the Queue whose attribute "Move target" contains the same value. This attribute contains the string "Start" like the corresponding attribute defined in the SAP Receiver / Sender template. The tool/material method "state constructed" can only be started by the Agent if the attribute "Move target" of the SAP Task template and the SAP Receiver / Sender template contain the same string (in the Ready-To-Use Setup the string "Start" is defined). Otherwise the tool/material method "state constructed" would be locked via the match criterion **AGENT(MUST_MATCH)** defined in the field "Type property" of the attribute "Move target" of the SAP Task template.

Connection ID (ConnID)

The Connection ID is required by the SAP Receiver for connection with Tasks. The SAP Adapter fills this attribute with the value of the **parameter CONNID** as the attribute contains the match criterion **SAPXOM(CONNID)** in the "Type Property" field.

Link Children (LinkChildren)

Currently not in use, leave unchanged (by default set to *).

Reply Message Group (RMG)

Reply Message Group ID provided by SAP for internal use thus Adapter can communicate with SAP. knows Name of the Local Output Management System, required to let SAP Receiver / Sender know which call-back rules should be uses for the related SAP Task. The SAP Adapter fills this attribute with the value of **option EG** of the **SUBMIT command** as the attribute contains the match criterion **SAPXOM(EG)** in the field "Type property".

Match Criterion (MatchCriterion)

As the attribute "Match Criterion" contains the "Type Property" **SAPXOM($MATCH_RECURSIVE$,CMD)** the SAP Adapter will compare the contents of **option CMD** sent by the **SUBMIT command** with the contents of this attribute to decide whether the Task template may be instantiated or not.

Spool ID (SPOOLID)

The SAP Adapter fills this attribute with the value of **option EI** as the attribute contains the match criterion **SAPXOM(EI)** in the field "Type Property".

Destination queue (DestinationQueue)

The attribute "Destination Queue" contains the name of the Queue the Agent will move the Task to. The SAP Adapter fills this attribute with the value of **option P** of the **SUBMIT command**, option P is assigned with the variable **&P** which contains the string entered in the field "Host printer" of the section *"Spool Administration : Output Device"*. As the attribute contains the match criterion **SAPXOM(P)** in the "Type Property" field.

At the first callback the SAP Receiver sends the Object ID of the SAP Task to SAP. This allows the SAP R/3 user to use the JobID to interrogate and identify the SAP Task in Papyrus Objects (this is done automatically by the Papyrus SAP Module in the background, no attribute or method is required).

Last state (Laststate)

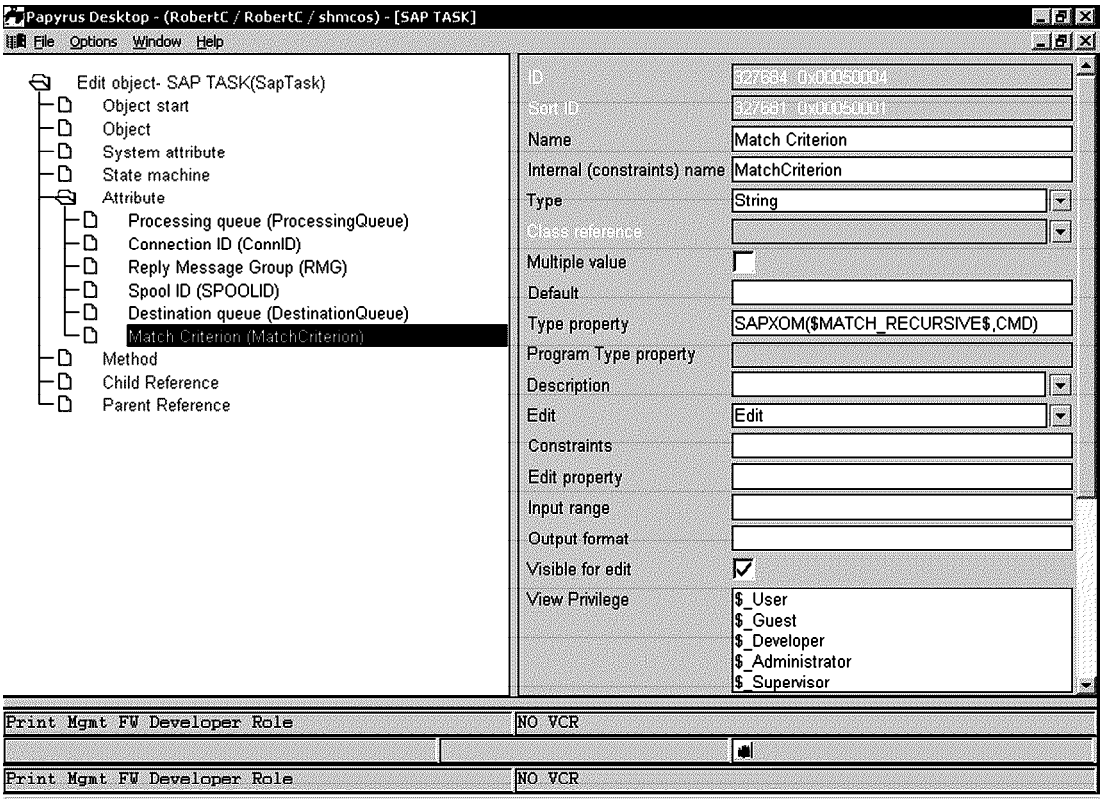
This attribute stores the last state of the SAP Task.

Last Commit state (LastCommitstate)

This attribute stores the last state of the SAP Task committed to SAP/R3. For the case of an abnormal program termination it is possible to compare the attribute "Last state", which stores the last state of the SAP Task and the attribute "Last Commit state", which is the last state of the SAP Task successfully received by SAP/R3. If both attributes are not equal after an abnormal program termination, SAP-XOM asks for the current state of the SAP Task or at least issues an error message.

SAP Title (SAPTitle)

The SAP Adapter fills this attribute with the value of **option T** (SAP-user title of the spool / output request from SAP entered by the SAP user) as the attribute contains the match criterion **SAPXOM(T)** in the field "Type property".



16.2.4 Methods of the SAP Task

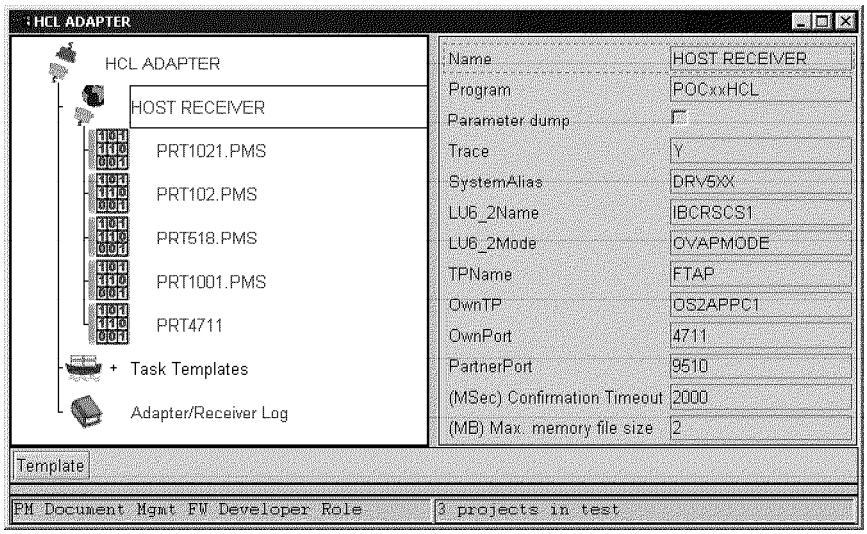
state constructed

The Material method "state constructed" is stared by the Agent together with the Tool method of the same method internal name or method implementation name defined in the SAP Receiver / Sender. The objective is to let SAP know which Task processes the SAP job, otherwise no communication would be possible. The tool/material method "state constructed" takes the parameters which have been sent by SAP to the LOMS for the output request. LOMS generates a Reply Message Group ID, a Spool ID, a Connection ID, and Subtitle. In addition the tool/material method "state constructed" sends a message to Papyrus Objects that the Task has been successfully instantiated in SAP/R3.

Parameter 1	ConnID=%ConnID%
Pre Constraint	EngineStatus==15

17.0 Host Client Receiver

The Host Client Receiver is an interface to the Papyrus Host running on a mainframe. The data file (AFP, PDF, etc.) and the control file are sent by Papyrus Host to the Host Client Receiver and the data is passed to its Adapter to instantiate the appropriate Task template. The required resources are taken from the Resource Collection as they are not sent by Papyrus Host via the Papyrus Host Client (appc.exe). On a mainframe with z/OS Unix (USS), a Papyrus Objects Kernel can be installed to run the Host Receiver directly on the mainframe, otherwise a Papyrus Objects Kernel on a remote machine (e.g. Windows, Unix, etc.) has to have a Papyrus Objects Kernel installed to run the Papyrus Host Receiver and receive the JCLs from the Papyrus Host on the mainframe.



17.1 Components of the Host Client Receiver

17.1.1 Attributes of the class "HOST RECEIVER"

Program	Executable behind the Host Client Receiver. As the Host Client Receiver is available for various platforms, the identifier POCxxHCL is defined, thus the system will retrieve path and filename from the appropriate Program object.
Parameter dump	If checked, the received parameters are output in the message panel.
Trace	Enables detailed trace messages.
SystemAlias	Alias or IP address of the machine where the Host Client Receiver is running.
LU6_2Name	TCP/IP name or IP address of the z/OS host.
LU6_2Mode	Log mode table entry name to be used for the z/OS host.
TP name	Name of the z/OS transaction program.
Own TP	Name of the fixed PC transaction program "OS2APPC1".
Own port number	Specify the TCP/IP server port number of the machine where the Host Client Receiver is running.
Partner port number	Specify the TCP/IP server port number of the z/OS host.
(MSec) Confirmation timeout	Specify a timeout that defines how long the Host Client Receiver has to wait for a confirmation after it has send a message to its Adapter. If this timeout is exceeded, an error will be returned to the z/OS host.
(MB) Max memory file size	Specify a limit for the size of the job data that the Host Client Receiver may pass to its Adapter via the memory. Job data exceeding this limit will be automatically stored on the hard disk and the Receiver passes a file pointer to its Adapter instead.

18.0 Papyrus Type Manager DB

Type Manager DB is used to access a database server. A simple IMPORT command will extract the table definitions (where possible) and create class definitions in the Papyrus Repository. It is even possible to do multiple joints (e.g. get all addresses from a general table but only those related to a specific company). With these class definitions, the Developer or Administrator can now define business object templates that represent the business objects. One row of the DB would typically represent one object instance. In this definition, the fields of the tables are mapped to the business object defined. Also, generic query templates can be defined. Queries produce a list object that is a table extract of the database.

18.1 Type Manager DB for standard SQL databases

Type Manager DB maps a customer generic database (or more than one) into OMS objects to grant queries for particular objects matching pre-defined criteria. In the first step we consider an Oracle Database (version 8.0.5). Type Manager DB can processes arbitrary data structures, i.e. the user is not limited to the ISIS database structure (c.f. PAR and PAS interface).

The method **"ImportDataStructure"** automatically generates classes in the parent class **Data Entities**.

The prerequisite is to have an Oracle client installed on the local machine (or the Domain Controller) and configured for the service (DBName) that should be used. Every customer table is mapped in an OMS class with an *ImportDataStructure* function retrieving the information by the Oracle System tables.

These information are about all tables owned by the database user specified in OMS *TypeManagerDB instance*.

18.2 class Type Manager DB

The class TypeManager DB contains information about the database in its attributes and has methods to connect and shutdown the Database. All methods are implemented in OMSDB.DLL.

states

InActive	The TypeManagerDB is not connected to the database. This is the initial state (state machine status) and in it the user can only activate the method "Connect" which changes the current Status to "Active".
Active	The TypeManagerDB is connected to the database. In this state the user can activate the Shutdown method that changes the current status back to InActive.

attributes

DBName	Database name to which the Type Manager DB connects.
DBUser	Login name
DBPasswd	Password
DBBrand	Database brand name such as Oracle 8.0.5., MSSQL Server 11.0 etc.
Trace	Checkbox field to active the trace messages containing the query syntax in the kernel window.

Note The messages are written into the kernel window where it is possible to copy the query syntax for pasting it into the SQL worksheet as the lines of the Desktop message window cannot be copied.

methods

Connect	This method establishes a connection with the service specified in the user schema (DBName, DBUser, and DBPasswd). After the connection the user can perform actions (import data structure, execute queries, updates and deletions) on the database until the shutdown occurs.
Shutdown	This method ends the connection with the specified user schema. After running this method the user cannot perform any action on the database until the connection is reestablished.
ImportDataStructure	This method reads customer Database information and creates a class for every table owned by the user specified by DBUser/DBPasswd on database DBName. It puts newly created classes as children of Database Entities class.

RunQueryDB	This method is the Tool part (Tool/Material) to execute a query on the database dynamically created from the information in Material object. It creates instances with the result values.
SetattributeToDB	This method is the Tool part (Tool/Material) to execute an update on database dynamically created from the information in Material object. It updates only one field at a time that isn't involved in a join.
DeleteFromDB	This method is the Tool part (Tool/Material) to execute a delete on database with criteria dynamically created from the information in Material object. It deletes only data that are not involved in a join.

18.3 Type Manager DB Example

Instantiate the **class Type Manager DB** on the local node and enter the appropriate user schema according the following example:

DBName	p01
DBUser	p01
DBPasswd	p01

Right click the new instance of the Type Manager DB and select *Connect* from the context menu or click the respective button in the toolbar, to establish a connection. If the connection works the message *"Successfully connected to db ..."* will appear.

Right click the Type Manager DB instance and select *ImportDataStructure* from the context menu or click on the respective button in the toolbar, to import the tables as child objects in the **class "Database Entities"** in the Repository. If the import was successful the message *"User tables successfully imported from db"* and a message about the number of tables imported will appear. All imported table columns are represented as attributes of the child classes in the class **Database Entities**. The notation is *TableName.TableColumn* and this is the "Constraint name" that must be maintained like this.

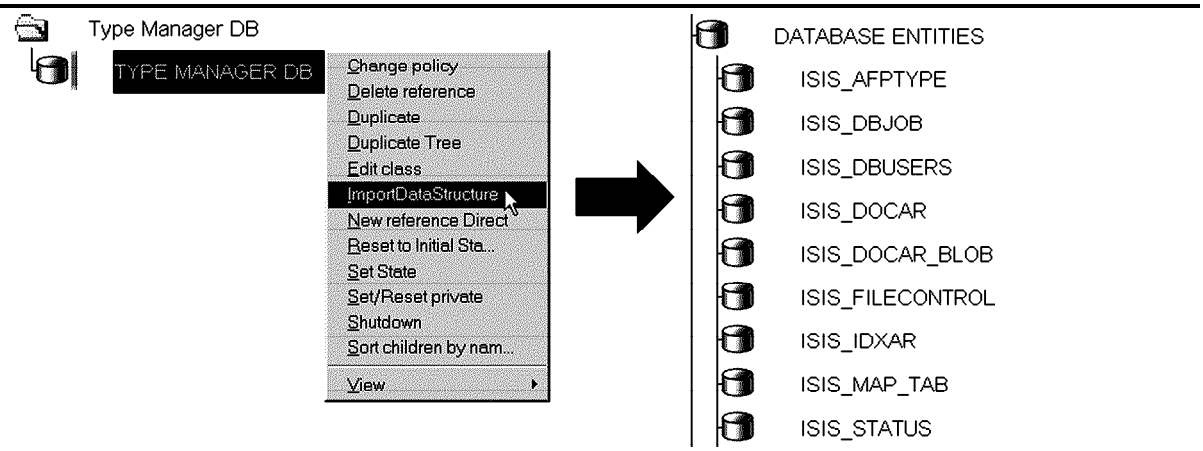


Figure 12: Execution of the method "ImportDataStructure"

19.0 Type Manager LDAP

The Type Manager LDAP can be used to authenticate users externally via the data in a LDAP directory. By this it is possible to centrally manage access rights for all users and systems of an organisation. Although the Type Manager LDAP can be used to import any other kind of data out of a LDAP directory, its main purpose is the external user authentication.

Main features of the Type Manager LDAP

- Importing authentication data (user name) from the LDAP directory to allow the authentication of users in Papyrus Objects. This is accomplished by adding instances of "User external", "Role external" and "Policy external" which are the counterparts to "User", "Role", and "Policy" used for the internal authentication.
- Authenticate users externally during the login process by accessing the LDAP directory. During the login process itself and everytime a view is opened the password of the external user is checked against the LDAP server. A password entered in the attribute "Password" by the user will be ignored during authentication.

During the import of authentication data the following tasks are possible:

1. Creating, updating and deleting of external Roles / Users / Policies in the Role / User / Policy collection. External Roles / Users / Policies which are not present in the respective Collection are created, external Roles / Users / Policies which are already present are updated, and external Roles / Users / Policies which are not present in the LDAP directory are deleted.
2. Linking and unlinking of Roles to Policies and Roles to Users which are assigned to each other in the LDAP directory.

The Type Manager LDAP can only deal with instances and templates of the classes "User external", "Role external", and "Policy external". Only templates derived of these classes can be instantiated by the Type Manager LDAP and only instances derived from these classes can be deleted or modified by the Type Manager LDAP. Thus the Type Manager LDAP cannot delete instances derived from the classes "User", "Policy" or "Role" nor instantiate templates derived from these classes.

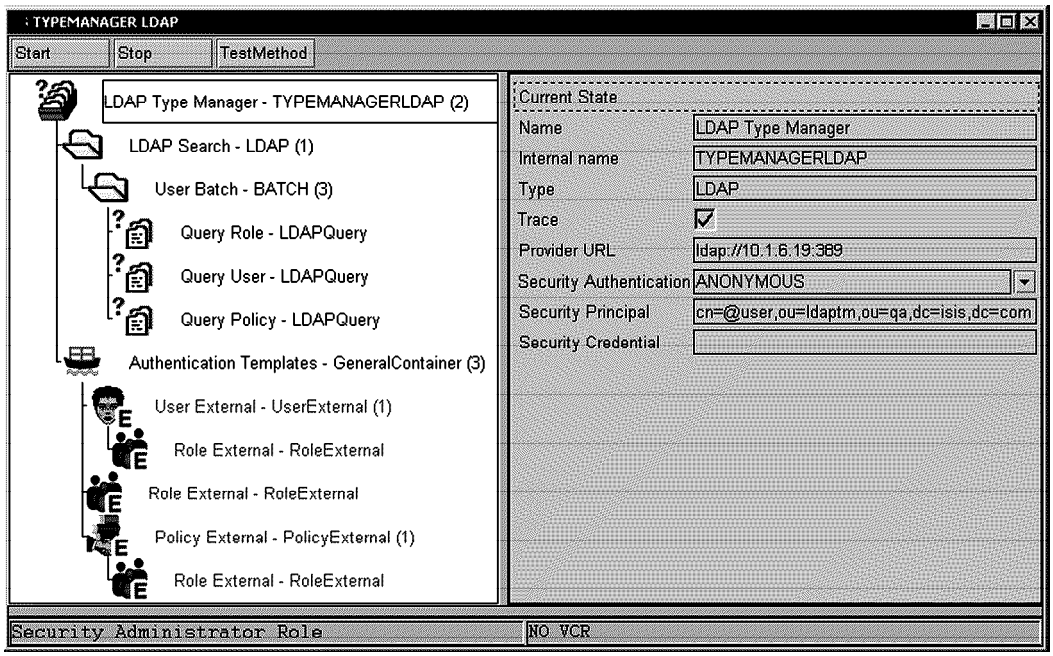


Figure 13: Typical setup for a LDAP Type Manager

19.1 Components of the Type Manager LDAP

The Type Manager LDAP requires the following components to be able to import external users:

Folder "LDAP Search"

This folder is derived from the class "FOLDER" with the Internal name "LDAP" containing Batches which are folders with the Internal name "BATCH". Batches are used to start one or more queries by using their Query objects. Query objects are instances derived from the class "LDAP Query" or "LDIF File" which store the information required to find the appropriate data in the LDAP server or to access the right LDIF file.

General Container "Authentication templates"

The General Container "Authentication templates" is derived from the class "GENERAL CONTAINER" and contains templates required by the Type Manager LDAP to instantiate external Users, Roles and Policies.

If the Type Manager LDAP should be run as a Kernel Service for authentication no children are required. For setups with a big number of users it is recommended to use two separated Type Manager LDAPs. One importing the user data at a certain interval (e.g. every 10 minutes) and the other one working as a Kernel Service (without children).

To run a Type Manager LDAP as a Kernel Service instantiate it below the Kernel Services Collection in the Storage and start the service. To enable Kernel Services the Scheduler must be started first.

19.1.1 Type Manager LDAP

The Type Manager LDAP can be used to authenticate users externally by accessing a LDAP directory during the log in process. Only users which have been derived from the class "User external" can be authenticated by the Type Manager LDAP via the LDAP server. The Type Manager LDAP can be used to import authentication data from a LDAP directory to create external Users, Roles, and Policies based on it. Furthermore it is possible to import the external Users during the authentication process by using the UserBatch, which will be executed at every login of an unknown and external user.

19.1.1.1 States

Inactive	The Type Manager LDAP is inactive.
Active	The Type Manager LDAP is activated and ready for processing a batch or authenticate a User by accessing a LDAP directory
Error	An error occurred.
Running Batch	The Type Manager is busy processing a batch job.
Error Batch	An error occurred during the processing of a batch.
Starting	The Type Manager is Starting.
Error Starting	An error occurred during the start up of the Type Manager.

19.1.1.2 Methods

RunBatch Runs the defined batches manually. The following method parameters have to be specified:

`principal` ... String used to replace the variable @**user** in the Security Principal field during the authentication with the LDAP server.

`credential` ... Password required to authenticate to the LDAP server.

`batch` ... The visible name of the batch which shall be processed.

19.1.1.3 Attributes

Type	String used to identify the Type Manager LDAP. Match criteria starting with this string are applied to the Type Manager LDAP with the same string in its attribute "Type".
Trace	The following options are available to output a trace: `Silent` = No trace information will be logged `Errors` = Errors will be logged `Errors and warnings` = Logs error and warning messages `Everything` = Logs the whole status information
Provider URL	URL of the LDAP Server e.g.: ldap://ldap.myorg.com
Security Authentication	Select which method (ANONYMOUS, SIMPLE or SASL) should be applied to check the authentication of an user via the LDAP protocol.
Security Principal	The full domain name of the user who shall be authenticated. The variable @**name** will be replaced by the login name during the process of accessing the LDAP server.
Batch Principal	The full domain of the principal (user) required to access the LDAP directory via batch (valid for all batches).
Batch Credential	The password required to access the LDAP directory via batch (valid for all batches).
Batch run time [s]	Defines the interval in seconds that have to elapse until the Batch "AutoBatch" is executed again (e.g. every ten seconds). If this attribute contains the value 0, the Batch "AutoBatch" will not be processed at all.

19.1.2 LDAP Query

Instances of this class can be added to a Batch Folder of the Type Manager LDAP to define queries directed to the **LDAP server**. The Type Manager LDAP will then instantiate the appropriate Authentication templates and fill their attributes with the collected information.

19.1.2.1 States

No states defined.

19.1.2.2 Methods

No methods defined.

19.1.2.3 Attributes

Name	Visible name which should describe the process performed by the query.
Internal Name	Do not change this identifier!
Query Number	The sequence of processing the queries inside a batch is determined by the attribute "Query Number", i.e. the Query with the lowest number will be processed first.
Base	This attribute determines the search base. A query can only access the base object and all objects referenced below the base.
Scope	The attribute Scope determines how many levels below a base object can be accessed by the query. The following options are available **LDAP_SCOPE_BASE** ... Only the base itself can be accessed by the query **LDAP_SCOPE_ONELEVEL** ... Only the first level below the base can be accessed by the query. **LDAP_SCOPE_SUBTREE** ... The whole subtree below the base object can be accessed by the query.

Filter	Expression which filters the relevant objects. This expression is processed by the LDAP server directly and has to comply with the LDAP query syntax. In the synopsis below the operator "**|**" (= logical OR), "**&**" (= logical AND), and "**!**" (= logical NOT) can be used.
	`(Operator(attribute=value)(attribute=value)...)`
	
	In the example below all objects derived from the LDAPclass **isisPolicy** or derived from the LDAPclass isisPolicyRole are included in the search:
	`(|(objectclass=`**isisPolicy**`)(objectclass=`**isisPolicyRole**`))`

attributes	List of attributes of a LDAP object which shall be returned by the LDAP server. If this field is left blank, all attributes of the query result will be returned.
Placement	Specifies the location of the instances which should be synchronised (e.g. to update an User external enter the Object ID of the User Collection).

19.1.3 LDIF File

Instances of this class can be added to a Batch folder of the Type Manager LDAP to define queries directed to a **LDIF file**. The Type Manager LDAP will then instantiate the appropriate Authentication templates and fill their attributes with the collected information.

19.1.3.1 States

No states defined.

19.1.3.2 Methods

No methods defined.

19.1.3.3 Attributes

Name	Visible name which should describe the process performed by the query.
Internal Name	Do not change this identifier!
Query Number	The sequence of processing the queries inside a Batch folder is determined by the attribute "Query Number", i.e. the Query with the lowest number will be processed first.
Path and filename	Enter path and filename of the ldif file (e.g. C:\LDAP\Users.ldif).
Placement	Specifies the location of the instances which should be synchronised (e.g. to update an User external enter the Object ID of the User Collection).

19.1.4 Batch folder

The Batch Folder is an instance derived from the class "Folder" with the Internal name "Batch".

The following batch folders are predefined:

User Batch	The queries of this batch will be started when the Type Manager LDAP works as a Kernel Service and an external or unknown user tries to log in.
Auto Batch	The queries of this batch will be started automatically when the attribute "Batch run Time" is set to a value greater than 0. The queries of this batch will be started every number of seconds specified via the attribute "Batch run Time".
MyBatch	This batch will be started when starting the method "RunBatch".

The user can add additional Batch folders with arbitrary Visible names and the Internal name "Batch". To call the queries stored in the Bach folder additional methods can be created (available method internal names are: RunBatch1, RunBatch2, RunBatch3, RunBatch4).

19.1.5 User External

Class User External used by the Type Manager LDAP to import and update Users with a LDAP directory. The User External can be seen as a extended User. The Type Manager LDAP uses this class instead of the class "USER" as it provides additional attributes required by the update mechanism. As instances of this class have been created automatically and will be updated by the Type Manager LDAP, **NEVER** change the instances in Papyrus Objects but modify the user data of imported users in the LDAP directory instead!

19.1.5.1 states

No state machine defined.

19.1.5.2 Methods

Contains inherited methods only.

19.1.5.3 Attributes

Name	Login name of the User (filled in by the Type Manager LDAP).
Internal Name	Unique name generated according the rules of Papyrus Objects internal names. (filled in by the Type Manager).
Password	Password of the user.
User Name	Full Name of the User (has no effect).
Language	Language of the user (has no effect).
Telephone number	Telefone Number of the user (has no effect).
Fax number	Fax number of the user (has no effect).
E-Mail	E-Mail of the user (has no effect).
Department	Department of the user (has no effect).
Location of user	Location of the user (has no effect).
ditinguished Name	Location of the user data of this particular user inside the LDAP directory (filled in by the Type Manager).
Common Name	common name of the user inside the LDAP directory (filled in by the Type Manager).
Match Criterion	By default the LDAP object class is the match criterion. To specify a user defined match criterion, change the field "Type Property" in the template.

19.1.6 Role External

A Role is a collection of privileges. Roles are used to define groups of users. The desired Privileges are selected from the attribute "Privilege".

Role External is a Role which has been defined by importing Roles from a LDAP directory. To import Roles from a LDAP directory the Type Manager LDAP is required.

The Role External is a full usable extended Role which enables the Type Manager LDAP to automatically update with a LDAP directory.

As instances of this class have been created automatically and will be updated by the Type Manager LDAP, **NEVER** change the instances in Papyrus Objects but change the user data in the LDAP directory instead!

19.1.6.1 states

No state machine defined.

19.1.6.2 Methods

Contains inherited methods only.

19.1.6.3 Attributes

Name	Visible Name.
Internal name	Internal name for the object. This identifier must be unique.
distinguished Name	The full domain name of the Role External inside the LDAP directory.
unique Name	Should be identical with the Internal Name.
Match Criterion	Corresponding LDAP object class which must be defined in the template of the Role External.
Privilege	Priviledges assigned to this Role.

19.1.7 Policy External

Policy External is a Policy which has been defined by importing Policies from a LDAP directory. To import Policies from a LDAP directory the Type Manager LDAP is required. The Role External is a full usable extended Policy which enables the Type Manager LDAP to automatically update with a LDAP directory. As instances of this class have been created automatically and will be updated by the Type Manager LDAP, **NEVER** change the instances in Papyrus objects but modify the external policies in the LDAP directory instead!

19.1.7.1 States

No state machine defined.

19.1.7.2 Methods

Contains inherited methods only.

19.1.7.3 Attributes

Name	Visible Name.
Internal name	Internal name for the object. This name must be unique.
distinguished Name	The full domain name of the Policy External inside the LDAP directory.
unique Name	Should be identical with the Internal Name.
Match Criterion	Corresponding LDAP object class which must be defined in the template of the Role External.

19.1.8 Group External

A Group External is a collection of Privileges imported from a LDAP directory. It is used to define a group of users. The Privileges of a Group External are stored in the attribute "Privilege".

Contrary to Roles, a User can belong to more than one Group at the same time. This is comparable to a user who has more than one active Role in the same Session. Group External is a Group which has been defined by importing Groups from a LDAP directory. To import Groups External from a LDAP directory the Type Manager LDAP is required.

As instances of this class are created automatically and will be updated by the Type Manager LDAP, **NEVER** change the instances in Papyrus Objects but change the group data in the LDAP directory instead!

19.1.8.1 states

No state machine defined.

19.1.8.2 Methods

Contains inherited methods only.

19.1.8.3 Attributes

Name	Visible Name.
Internal name	Internal name of the object. This identifier must be unique.
distinguished Name	The full domain name of the Group External inside the LDAP directory.
unique Name	Should be identical with the Internal Name.
Match Criterion	Corresponding LDAP object class which has to be defined in the template of the Group External.
Privilege	Priviledges assigned to this Group.

19.2 Match Criteria for the Authentication templates

The following extended variants of Match Criteria can be used in the Authentication templates (User external, Role external, and Policy external) thus the Type Manager LDAP is able to decide which of these templates should be instantiated and which values received from the LDAP server or the LDIF file should be written into the respective attributes:

LDAP($MATCH_LOCAL$,*LDAP_attribute*,*Placement*)

If the value of the attribute equals the value of the LDAP attribute the Type Manager LDAP adds an instance to the specified object. Placement is the object ID of object where the instance should be placed (e.g. via the match criterion LDAP($MATCH_LOCAL$, objectclass, 0.0.0.18) the Type Manager LDAP would create an instance in the User Collection which has the Object ID 0.0.0.18 if the attribute contains the same value as the LDAP attribute).

LDAP(*LDAP_attribute*)

The attribute takes the value of the LDAP attribute.

LDAP($ISIS_objectNAME$,*LDAP_attribute*)

The attribute, the Visible name, and the Internal name take the value of the LDAP attribute (e.g. LDAP($ISIS_objectNAME$,cn))

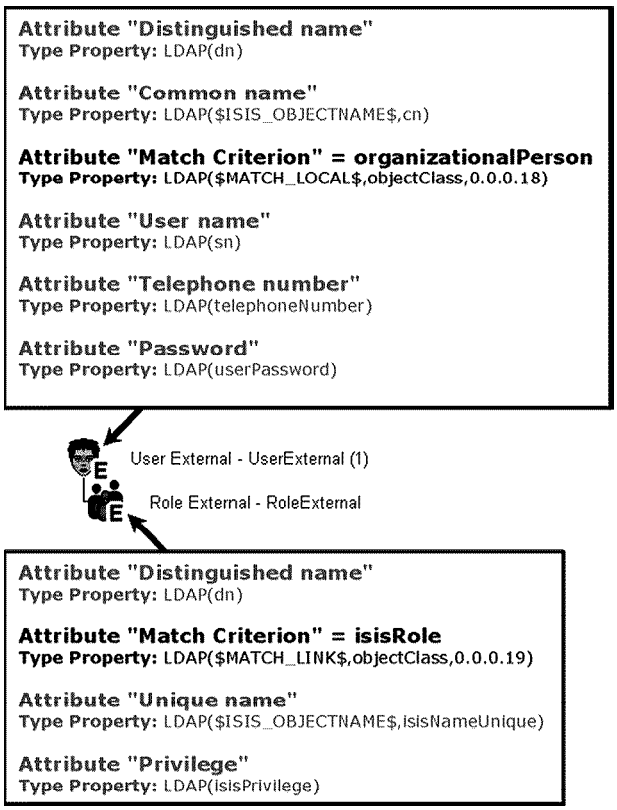
LDAP($MATCH_LINK$, *LDAP_attribute*, *Placement*)

If the value of the attribute equals the value of the LDAP attribute the Type Manager LDAP adds a reference to the specified object. Placement is the object ID of object where the reference should be placed (e.g. via the match criterion LDAP($MATCH_LINK$, objectclass, 0.0.0.19) the Type Manager LDAP would create a reference in the Role Collection which has the Object ID 0.0.0.19).

19.3 Setup of the Authentication templates

To setup the Authentication templates, add templates of the classes "User external", "Role external", and "Policy external" in the General Container of the Type Manager LDAP. Define a match criterion of type LDAP(LDAP_attribute) for each attribute that should retrieve the value from the respective LDAP attribute. Use a match criterion of type $MATCH_LOCAL$ to let the Type Manager LDAP instantiate this Authentication template if the attribute contains the same value as the corresponding LDAP attribute. Add a match criterion of type $MATCH_LINK$ to let the Type Manager LDAP create a new reference if the attribute contains the same string as found in the related LDAP attribute.

The following example shows which match criteria have to be defined in a template for an external User containing an external Role or internal Role, thus these Authentication Tempaltes match the setup inside the LDAP directory described in the next sub chapter:

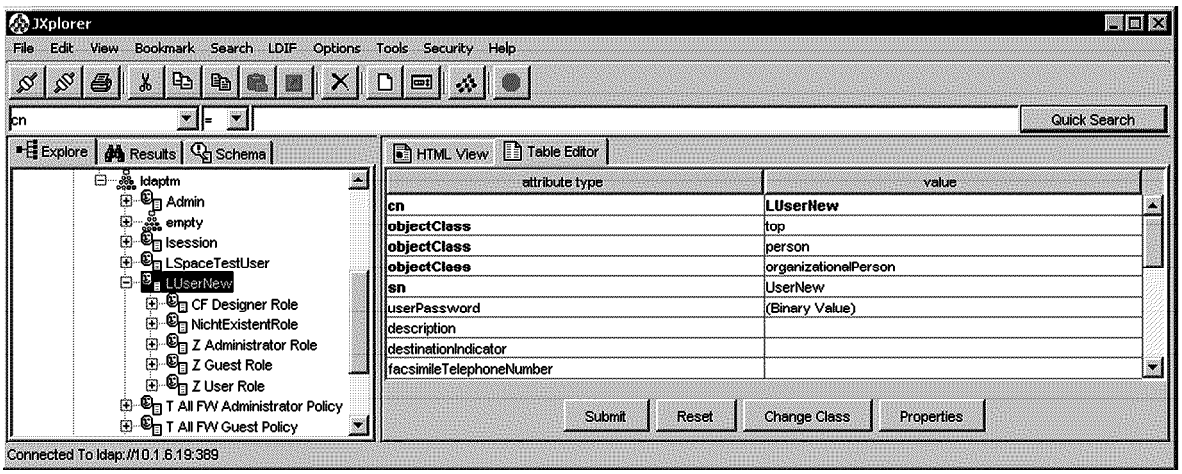


19.4 General setup inside the LDAP directory

Create the schema isis.schema on the LDAP server and add the following objectclasses.

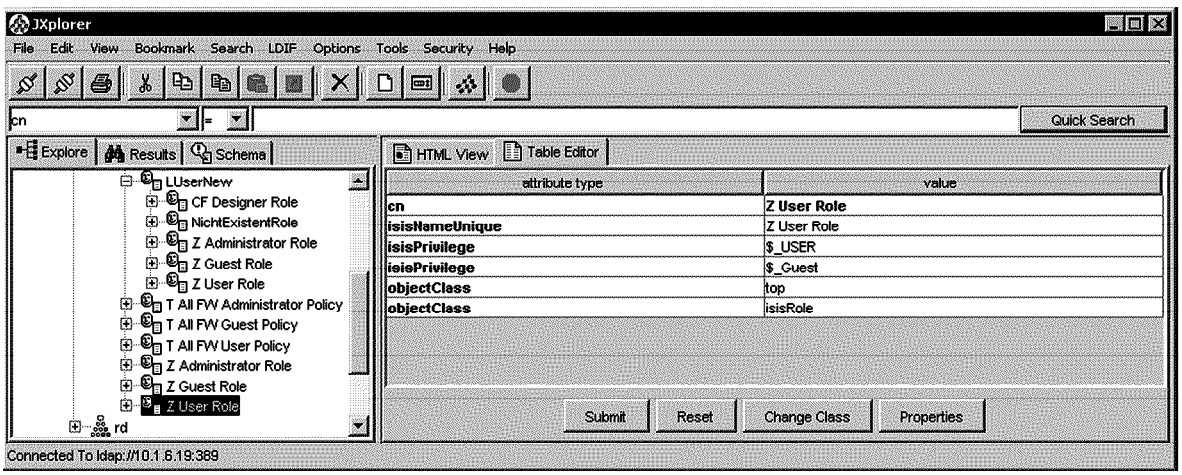
User external

The User external should be derived from the LDAP class ***organisationalPerson*** or any other class containing a password attribute. Via ***cn*** the login name has to be specified.



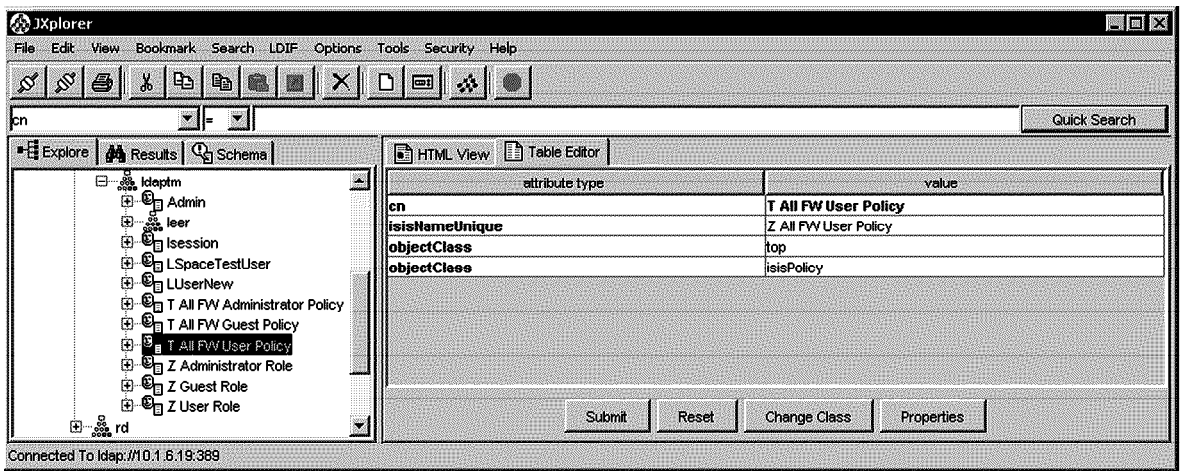
isisRole

The LDAP objectclass ***isisRole*** is used to define a Role external inside LDAP, ***isisPrivilege*** contains all Privileges of the Role, and ***isisNameUnique*** contains the Visible name of the Role external.



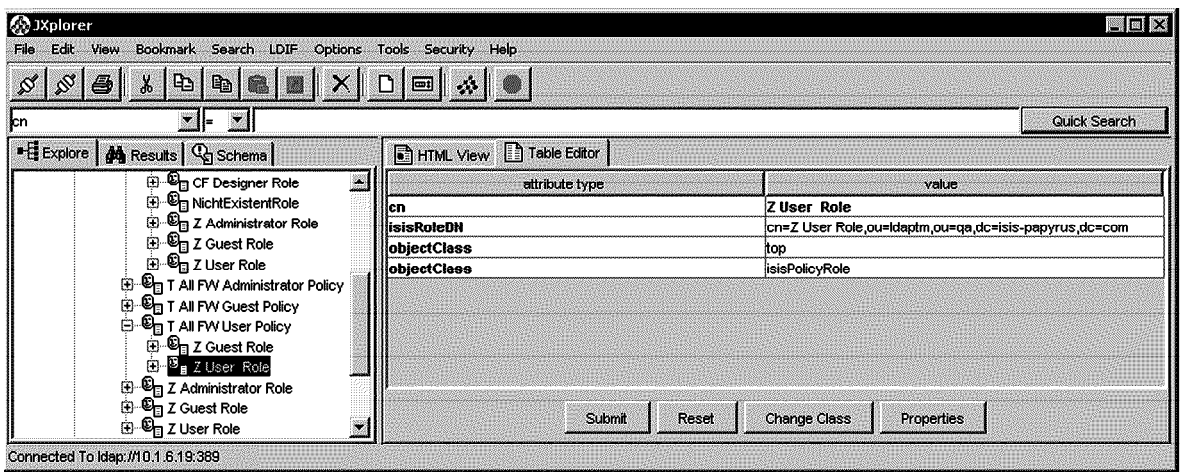
isisPolicy

The LDAP objectclass ***isisPolicy*** is used to define a Policy external inside LDAP and ***isisNameUnique*** is the Visible name of the Policy external.



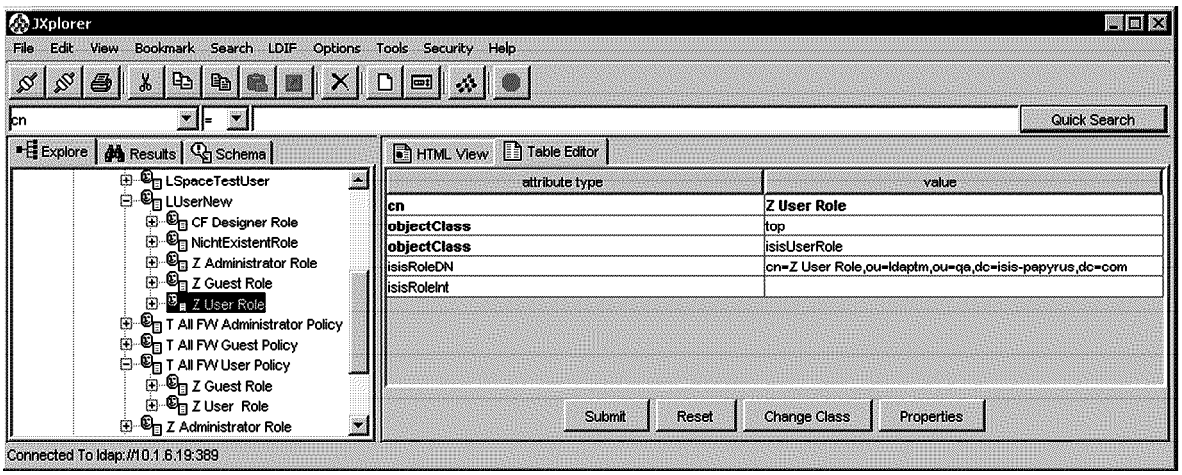
isisPolicyRole

The LDAP objectclass ***isisPolicyRole*** is used to define a reference between a Policy (internal or external) and a Role external inside LDAP. The LDAP attribute ***isisRoleDN*** comprises the full LDAP ***dn*** of the ***isisRole*** inside LDAP.

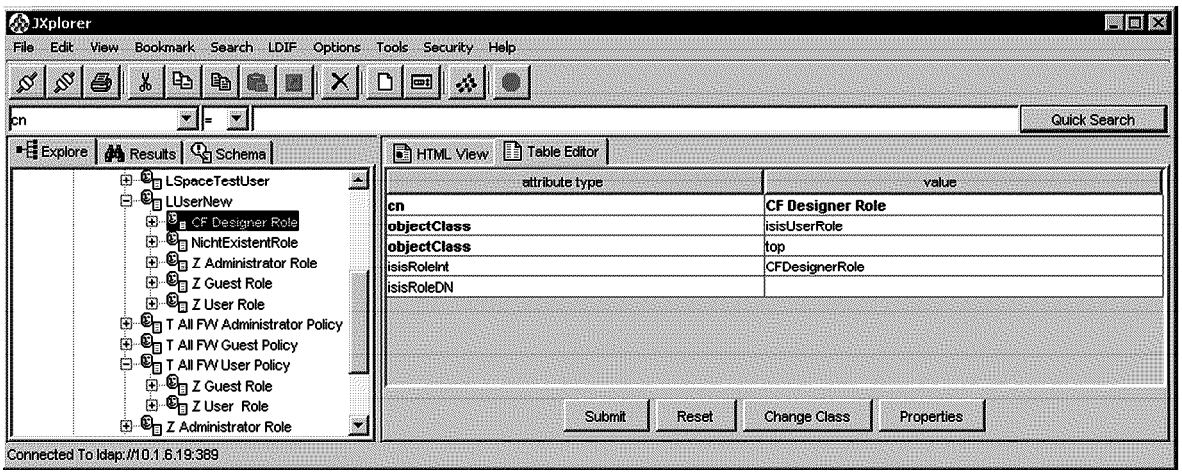


isisUserRole

The LDAP objectclass ***isisUserRole*** is used to define a reference between a User external and a Role (internal or external) inside LDAP. The LDAP attribute ***isisRoleDN*** comprises the full LDAP ***dn*** of the ***isisRole*** inside LDAP.



The LDAP attribute ***isisRoleInt*** comprises the full Internal name of an internal Role defined in Papyrus Objects.



Glossary

Adapter
An Adapter is used in combination with a Receiver for the automatic instantiation of templates based on match criteria. The name value pairs obtained from the Receiver are used to fill attributes and to decide if a template may be instantiated.

Administrator
The Role with the authority to authorize and constrain User access to objects.

AFP
Advanced Function Printing (or Presentation). IBM Printing Architecture.

AFPDS
Advance Function Printing Data Stream. AFP - Data Stream is the document file format of AFP.
It is also known as LIST3820 format in the OS/390 world. It is generated for example by DCF/Script or by Papyrus DocEXEC.

Agent
An automated user moving Tasks into the appropriate Queues based on conditions defined in Rule objects (to check attributes and states) and starting matching tool/material methods.

ASCII
American Standard Code for Information Interchange (Standard character array for letters and symbols)

attribute
Data about an object contained in the object.

Big Endian
Byte order in which multibyte numbers are stored on systems with Motorola processors. The high-order byte of the number is stored in memory at the lowest address and the low-order byte at the highest address.

Bridge
Used by the Papyrus Objects Kernel to start the DLLs behind general modules of Papyrus Objects (e.g. MQ Receiver) that are not part of Papyrus Server.

Child
A child class inherits all features of its Parent class. This concept allows the developer to reuse code and only extend the child classes with attributes and methods for more specific tasks.

Circular references
references pointing to themselves are called Circular references, they are used to design work cycles (e.g. Completed Queues containing a reference to the Distribution Queue, thus it is possible to restart a print job by changing its state and thus automatically move it back into the Distribution Queue on top). Circular references may cause a decrease in performance as they are still in the Object Space and monitored by the Scheduler although they are no longer visible on the Papyrus Desktop. Therefore it is adamant to delete Circular references properly by first deleting all references in the Sub Queues (e.g. Completed Queues containing a reference to the Distribution Queue) and thereafter the Queue on top itself (e.g. Distribution Queue that has been referenced in the Completed Queues). To purge already incorrectly deleted Circular references right click the Node and select the method "Check objects extended" in the context menu or click the respective button in the toolbar.

class
classes are building plans containing all common properties and methods that belong to an object. All classes reside exclusively in the Repository.

Client Node
A Client Node is a Node installed on a local workstation communicating with all other Client Nodes and the Domain Controller Node via TCP/IP, it allows the user to perform his tasks. Such a Node does not provide any services to other Nodes like the Domain Controller does. The graphical user interface, Papyrus Desktop, can be installed as well.

Clipboard
A method for moving objects from one Domain Server to another.

Collection
A special folder that holds groups of object such as Users, Roles, Nodes, etc....

Constraint
A condition which must be true in order to enable an attribute or a method.

CSV
Comma-separated values, a tabular data format where the fields are separated by a delimiter character (e.g. colon, semicolon, tab, etc.). Line data input files are CSV files used by Papyrus DocEXEC as input data (q.v. Line data input file).

DDE
Dynamic Data Exchange is a Microsoft Windows hotlink protocol (changes to the data made by one application appear instantly in the other's copy.) that allows application programs to communicate using a client-server model.

Denial-of-service attack
Attack on a computer system or network by overloading its resources.

Developer
A Role with the authority to create, delete and alter classes to develop and maintain applications

DLL
Dynamically Linked Library. Library containing procedures that can be shared between different programs at the same time (c.f. user32.dll containts all standard dialogs).

DocEXEC
The Papyrus Document Formatter.

Domain Controller Node
A Domain Controller is a Papyrus Objects Kernel running on a server that can be accessed by all Nodes via TCP/IP, it is used to manage all other Nodes in the Papyrus Objects Domain. The graphical user interface, Papyrus Desktop, can be installed as well.

EBCDIC
Extended Binary Coded Decimal Interchange Code

EXE method
EXE method calls procedures within an OMS enabled program, which use shared memory to communicate with the OMS Kernel.

Framework
A Framework is a set of classes and templates (optionally instances and references to other Frameworks) that embody an abstract design for a workflow. The System Framework, which is the basis of the whole system, and the Print Management Framework, which is the new graphical user interface of Papyrus WebControl, cannot be modified. All other Frameworks, like the Correspondence System (Document Framework), can be modified by the user.

GUID
Short for ***G****lobally* ***U****nique* ***Id****entifier*, a unique 128-bit number that is produced by the Windows operating system or by some Windows applications to identify a particular component, application, file, database entry, and/or user. GUIDs can be created in a number of ways, but usually they are a combination of a few unique settings based on specific point in time (e.g. an IP address, network MAC address, clock date/time, etc.), this way a GUID is unique worldwide and it might not happen that another computer produces the same GUID. A GUID is also used in a Windows registry to identify COM DLLs. Knowing where to look in the registry and having the correct GUID yields a lot information about a COM object (i.e., information in the type library, its physical location, etc.). Windows also identifies user accounts by a username (computer/domain and username) and assigns it a GUID.

Icon Collection
Contains all Icons and resides in the Storage Folder.

ICD
ISIS Control Data.

IMS
IBM's transactional and hierarchical database management system for critical on-line operational and e-business applications and data, enabling Information Integration, Management, and Scalability.

Inheritance
Each class is a separate module and has a position in a "class hierarchy". methods and attributes of a base class can be passed down the hierarchy
to child classes, this is called "inheritance".

instance
instances are objects derived from templates or directly from classes and used as an interface to the user containing only the methods and attributes enabled by the Administrator. Instances are the object type representing entities for the work utilized by processes or the user and usually carry no definitions like classes and templates.

Instantiation
Instantiation is the idea of creating a new object based on the design of a source object.

IPDS
Intelligent Printer Data Stream.
A standard IBM hardware printer protocol. IPDS and AFPDS are very similar internally, but should not be confused. IPDS is a protocol and AFPDS is a file format.

IPAS
All Nodes require for startup a product authorization managed by the *ISIS Product Authorization Service (IPAS).* In most cases the Domain Controller Node is used for this service but it is also possible to use a separate Node for the IPAS, running on another or the same server.

Kernel
see "Papyrus Objects Kernel"

LDAP
Lightweight **D**irectory **A**ccess **P**rotocol is an open protocol which makes it possible for almost any application running on virtually any computer platform to obtain directory information, such as email, addresses and public keys (c.f. ActiveDirectory on MS Windows 2000).

Library
Folder object where all templates are stored.

Line data input file
Line data input files are CSV files used by Papyrus DocEXEC as input data (q.v. CSV).

Little Endian
Byte order in which multibyte numbers are stored on systems with Intel processors. The low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address.

LOMS
A **L**ogical **O**utput **M**anagement **S**ystem represents a specific function of an external output management system in the R/3 System.

Man-in-the-middle attack
A computer security breach in which a malicious user intercepts - and possibly alters - data traveling along a network.

Material
A method type that is processed by some Tool method.

Material object
An object that has a Material method defined.

MetaClass
base class of all classes containing general methods that apply for all classes.

Modification Events
Modification Events are used only internally by programs, while Change Events can be used to trigger state transitions

method
The actions that can be performed by an object.

MQ Series
A messaging system for communication among programs by IBM.

Mutex
A mutual exclusion object that allows multiple threads to synchronise access to a shared resource. By this it is possible to define that a certain thread has to be finished until the next thread may take over.

Node
A Node consists of a Papyrus Objects Kernel and its related Object Space running on a machine which is connected via TCP/IP to the Papyrus Objects Domain (LAN or WAN). On one physical machine there may be more than one Node running simultaneously. The graphical user interface Papyrus Desktop is optional and is automatically installed if available for the operating system specified via the installation routine. Alternatively it is also possible to use a web browser, if a Papyrus WebPortal is installed on the Node whose Object Space should be accessed (thin client solution).

object-oriented programming (OOP)
object-oriented programming is a type of programming in which programmers define not only the data type of a data structure, but also the methods (=operations, functions) that can be applied to the data structure. This way, the data structure becomes an object that includes both data and functions.

objects
An object is the smallest functional unit in the Object Space. It is a data structure which consists of attributes (may be of different type), methods (=operations, functions), that can be applied to this or other data structures, states and their state transitions, describing the lifecycle of the object, and Parent / child reference definitions, that define for instance from which class an object must be derived that it may be added as child object and vice versa. Classes, templates, and instances are the three different types of objects.

object tree
The object tree in Papyrus Desktop displays all objects accessible to the user in a directory like structure. The parent child relationship is depicted by the tree, it either reflects the inheritance (classes in the Repository) or an arbitrary functional relationship (instances and templates). Objects in such a tree structure containing other objects are called parent objects, the objects they contain are called child objects.

Object Space
The Object Space is the location where the objects (data units), logs, and settings of Papyrus WebControl and WebRepository are stored (file system or database).

OCL
object Constraint Language

OMG
object Management Group, a consortium aimed at setting standards in object orientated programming.

OMS
object Meta System, the basis of *Papyrus Objects*

Papyrus Desktop
Papyrus Desktop is the graphical user interface to visualize the object Space, it is dependent on a Papyrus Objects Kernel to visualize objects on the local or on remote Nodes.

Papyrus Objects Kernel
The Papyrus Objects Kernel is the core system managing the Object Space and the communication with other Nodes.

Papyrus Version Control
Papyrus object offers an automatic version control for distribution and access to versions and variants of objects, this also applies to archived documents and resources.

Papyrus WebPortal
Web Portal is the thin client solution to the fat client Papyrus Desktop. By running a Papyrus WebServer on a Node, which turns it into a so called PPL Node, it is possible to use a web browser to access its Object Space similar to the Papyrus Desktop (fat client). By this no maintenance is required on the client workstations but via the web browser no Proxy objects are generated, thus the user is more dependent on the network than with Papyrus Desktop.

Papyrus WebServer
If a Papyrus WebServer is running on a Node it can be used as a PPL Node (Papyrus WebPortal, thin client solution), thus it is possible to use a web browser to access its Object Space similar to the Papyrus Desktop (fat client).

Permanent Running Mode
Papyrus DocEXEC task (e.g. started by host), loads all resources, runs until it finds a termination record in the DocDef, and then waits in standby for the next run.

OMS-aware
see "Papyrus Objects enabled"

OpenSSL
OpenSSL is an open source implementation of the SSL and TLS protocols. The core library (written in the C programming language) implements the basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

Papyrus Objects enabled
A program that is able to communicate with Papyrus Objects.

Parent
A Parent class is the base class to its child classes and is intended to define common attributes and methods that all its child classes share together.

PCL
Printer Control Language

PDF
Portable Document Format

Pipe
A pipe is a technique for passing information from one program process to another. Basically, a pipe passes a parameter such as the output of one process to another process which accepts it as input. The system temporarily holds the piped information until it is read by the receiving process.

Policy
Policies, like Users, are a collection of Roles used to assign authority (visibility) to objects.

PO_HTML
see "PPL files"

PPL
Papyrus Portal Language

PPL files
PPL files are HTML files with additional PO_HTML tags, which allow to use a web browser to display and modify the Object Space similar to the Papyrus Desktop (fat client). To be able to access the Papyrus Objects Domain via a web browser, a Papyrus WebServer must be running on a PPL Node. The Papyrus WebServer translates the PO_HTML tags within the PPL files to static HTML and JavaScript, thus the web browser on the client side is able to parse these web pages.

PPL Node
A Node that can be accessed via the web browser is referred to as a PPL Node.

Private object
An object that is only visible to a single User.

Privilege
Privileges are the base security items of Papyrus Objects. The system compares Privileges to determine if a given User may view or have access to an object.

Project Management instance
Each Project Management instance, which can be found in the Project Management Collection of the Storage, is a separate virtual development and test environment. All Projects attached to such a Project Management instance are a part of this virtual development and test environment. The Roles below a Project Management instance define which users are permitted to work with the Projects therein. A user can only access the Projects if his active Role can also be found in the respective Project Management instance.

Project instance
A Project is a collection of versions of classes and templates. Under Version Control a Project has to be defined to be able to create / modify classes or templates, create new versions of classes or templates, and duplicate a template or a tree of templates. By adding a Project to the Project Management instance, it becomes a part of this virtual development and test environment.

Proxy
A Proxy is a local copy of an object, the purpose is to reduce network traffic. Proxies have the same functionality the cash has for an internet browser.

PSF
Print Services Facility.
PSF/2, PSF/6000, PSF/400, PSF/VM, PSF/VSE, PSF/MVS are all IBM products for driving IPDS printers on various platforms.

RACF
Resource **A**ccess **C**ontrol **F**acility) is IBM's large system security product. It originally ran only under MVS but has since been ported to run under VM.

Receiver
A Receiver is used to receive messages and events outside Papyrus Objects passively, it passes these messages and events over to the Adapter for further processing.

Repository
Folder object where all classes are stored.

reference
A reference is a link to an object which can reside anywhere on a node being a part of a corporate Papyrus Objects domain. Each object in the Papyrus objects system is represented by an icon in the object tree. All Icons on the Desktop are basically references to objects (in other words visualizations of objects) which reside in the so called Object Space. One or more additional references may be added to any object.

Role
Roles are assigned with Priviliges that reflect the access and view rights for specific groups of users.

ROMS
A **R**eal **O**utput **M**anagement **S**ystem is an external Output Management System in an R/3 System.

Rule
Rules are used to define conditions based on attributes or states to trigger certain actions (e.g. change the value of an attribute, create a new reference).

SAP R/3
SAP R/3 is an integrated suite of applications for enterprise resource planning (ERP).

Session
Sessions define the contents of the Desktop (cf. User Profile) and are children to User objects. Every user has at least one Session that is created automatically if there is no Session object for a user.

Shared Memory
Memory in a computer shared by *Papyrus Objects* components for communications.

Shell method
method implementation type to lauch non-OMS-aware external executable files from the command line.

Socket
A socket is used as a communication interface to connect an application to a network protocol, data can be read from and written to the socket to directly communicate with the other application. The transport of the TCP/IP messages across the network is managed by the operating system.

state
Each object may have a set of states bound to it, this way it is possible to control processes as a state can be used to define which methods and attributes are currently available (e.g. only if an AFP has already been created the conversion into a PDF is possible).

SSL
Secure **S**ocket **L**ayer is a protocol designed by Netscape Communications Corporation to provide encrypted communications on the Internet. SSL is layered beneath application protocols such as HTTP, SMTP, Telnet, FTP, Gopher, and NNTP and is layered above the connection protocol TCP/IP. It is used by the HTTPS access method.

Starter
Used by the Papyrus Objects Kernel to start the DLLs behind the Papyrus Server Modules, which would otherwise be a task of Papyrus Server. Starter Modules are DLLs of Papyrus Server (e.g. File Receiver (ICD) is the Dispacher DLL of Papyrus Server).

state machine
Lists the states that an instance can have, and the Transitions between these states.

state transition
state transitions enable an object to change from one state to another. state transitions always move from their source state to their target state. If it is necessary for an object to move back and forth between two states then it will need two different Transitions.

Storage
A folder object where system relevant instances are stored.

TCP/IP
IP (Internet Protocol) specifies the format of packets, also called datagrams, and the addressing scheme. IP by itself is something like the postal system it deals only with packets. It allows to address a package and drop it in the system, but there is no direct link between the sender and the recipient. Most networks combine IP with the higher-level protocol TCP (Transmission Control Protocol), which enables two hosts to establish a connection to exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

template
A template is a constructor for an object. A group of objects sharing common attributes can be defined in a template. Once such a template is defined the only the default values need to be changed in the derived instance.

TLS
Transport **L**ayer **S**ecurity is a protocol that provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Tool method
A method type that processes certain kinds of Material methods.

Tool object
An object that has a Tool method defined.

Type Manager
A Type Manager is an interface which can access data stored outside Papyrus Objects actively (c.f. Receiver is passive). Type Manager DB enables access to certain database products without SQL, Type Manager LDAP can be used to obtain user authentications defined outside Papyrus Objects.

UML
Unified Modelling Language, used for modelling object orientaded systems.

VPN
Virtual **P**rivate **N**etwork uses access control, encryption, and other security mechanisms to ensure that only authorized users can access the network and that the data cannot be intercepted. VPNs enable companies to build a save network for their external agents using the infrastructure of the Internet.

Version Control
see "Papyrus Version Control"

WebServer
see "Papyrus WebServer"

XOSI Interface
Used by the Papyrus Objects Kernel to start the DLLs behind modules of all inbound products (e.g. Papyrus Capture Framework).

Zip File
Compressed archive file.

Index

$

$MATCH_DESTINATION$ 23

A

Adapter Alias 10
Adapter Project 6
Autostart on New 7

C

Confirmed_LinkRejected 6
Confirmed_LinkReopend 6
CopyFrom 51

D

Dispatch.ini <File_control> Parameter Summary 35
Dispatch.ini - Profile Description 25
Dispatch.ini - Sample Profile 35
DispatchFileName / 6

F

FAX Receiver / Sender 74
Factory Adapter 15
Factory Features of the Adapter 15
File Receiver 24
File Receiver and Papyrus Host 38
File mapping, PQL based 57

H

HTTP Receiver / Sender 102
Host Client Receiver 162

I

ICD Parameter File 40
ICD parameter length 24
INPUT_SIZE 6
Instance Alias 8
Interfaces 4

L

LDAP Type Manager 168

M

MQSeries Receiver 116
Match Criteria 11
Message Container 20
Message objects 20
Message template 21

P

POP3 Receiver 132
PPMF Parameter Summary 41
PPMF.INI 39
PQL based file mapping 57
Papyrus Query Language 17
Parameter Summary for the Modules 52
Project of Adapter 6

S

SAP Adapter 143
Source alias 8
Starting the ICD Receiver (Dispatcher) 35

T

Template Alias 8
Type (attribute of the Agent) 8
Type Manager 4
Type Manager DB 164
Type Manager LDAP 167
Typographic Conventions 3

U

Using Papyrus Server 40

X

XML Receiver 62